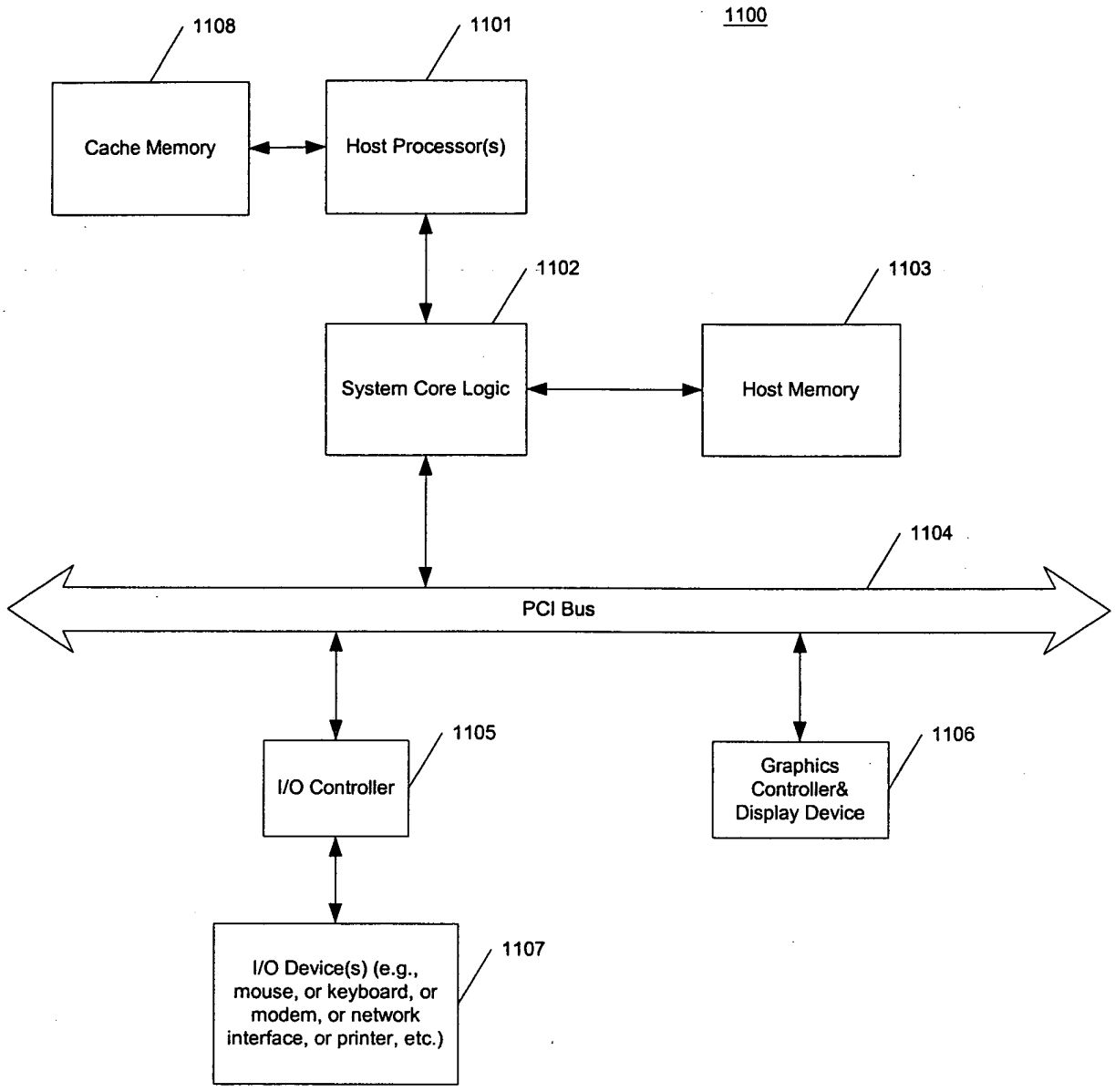

1100
1108
1101
Cache Memory
Host Processor(s)
1102
1103
System Core Logic
Host Memory
1104
PCI Bus
1105
1106
I/O Controller
Graphics Controller& Display Device
1107
I/O Device(s) (e.g., mouse, or keyboard, or modem, or network interface, or printer, etc.)

Figure 1

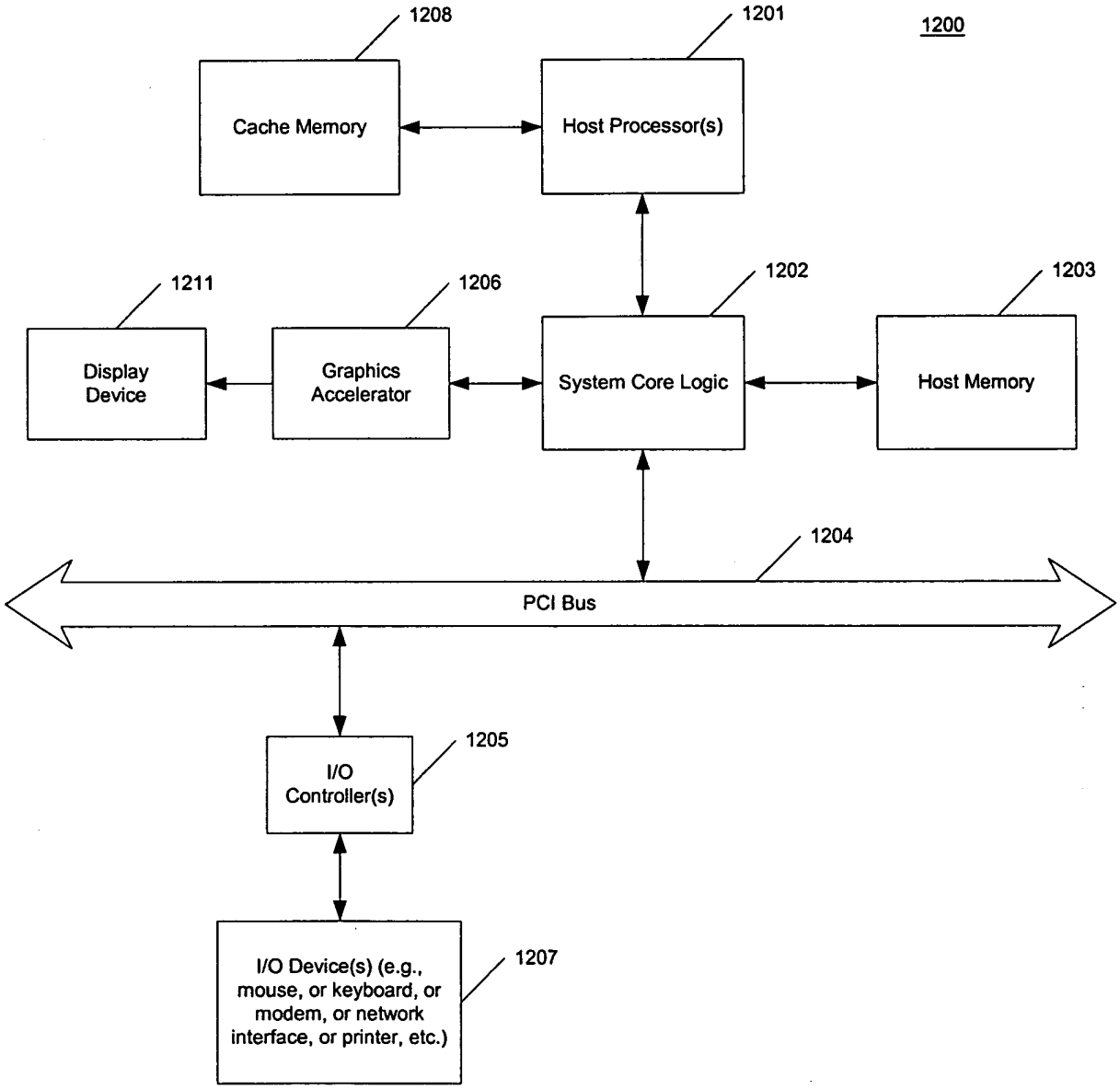

1208
1201
1200
Cache Memory
Host Processor(s)
1211
1206
1202
1203
Display Device
Graphics Accelerator
System Core Logic
Host Memory
1204
PCI Bus
1205
I/O Controller(s)
1207
I/O Device(s) (e.g., mouse, or keyboard, or modem, or network interface, or printer, etc.)

Figure 2

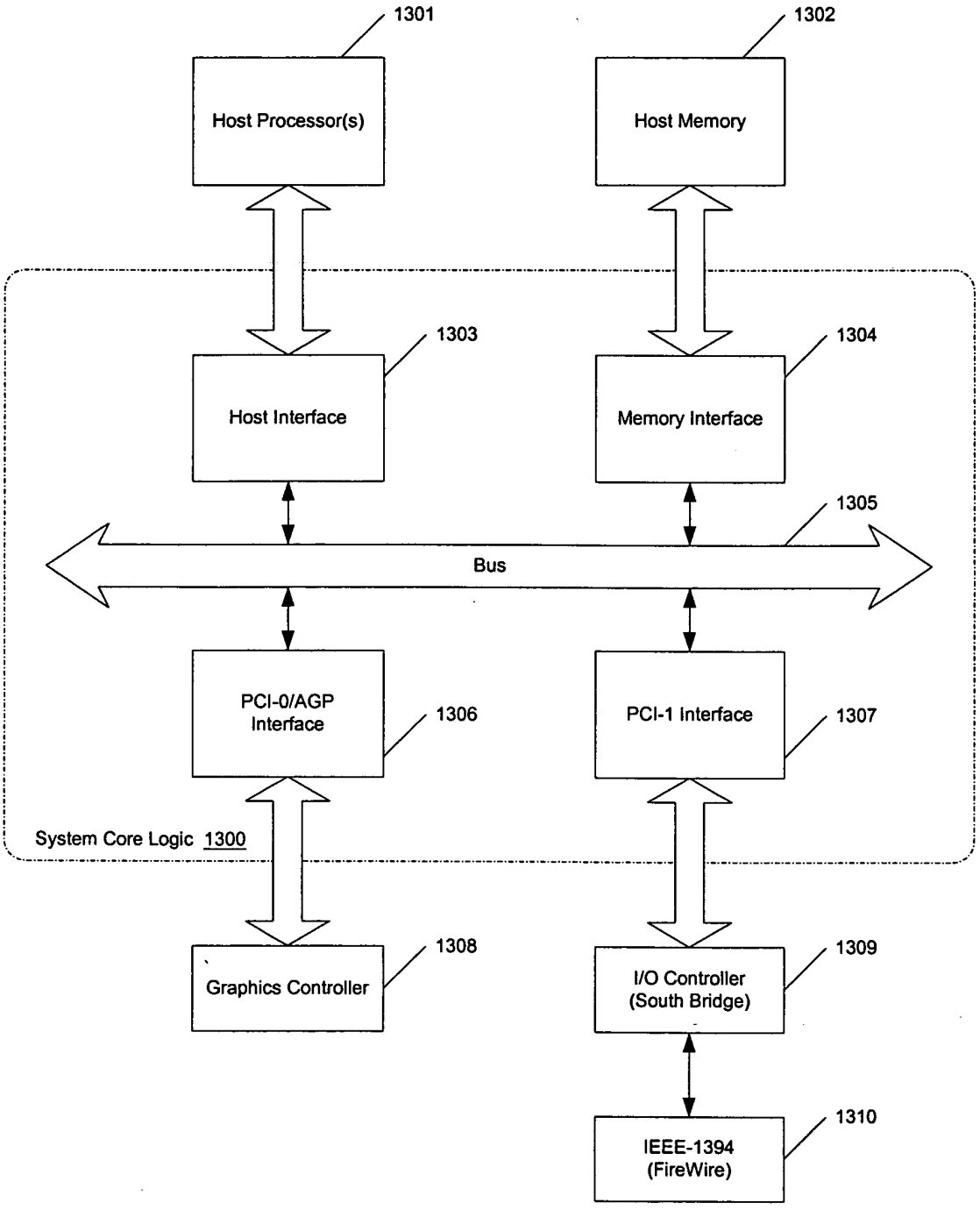

1301
Host Processor(s)
1302
Host Memory
1303
Host Interface
1304
Memory Interface
1305
Bus
PCI-0/AGP Interface
1306
PCI-1 Interface
1307
System Core Logic 1300
1308
Graphics Controller
1309
I/O Controller (South Bridge)
1310
IEEE-1394 (FireWire)

Figure 3

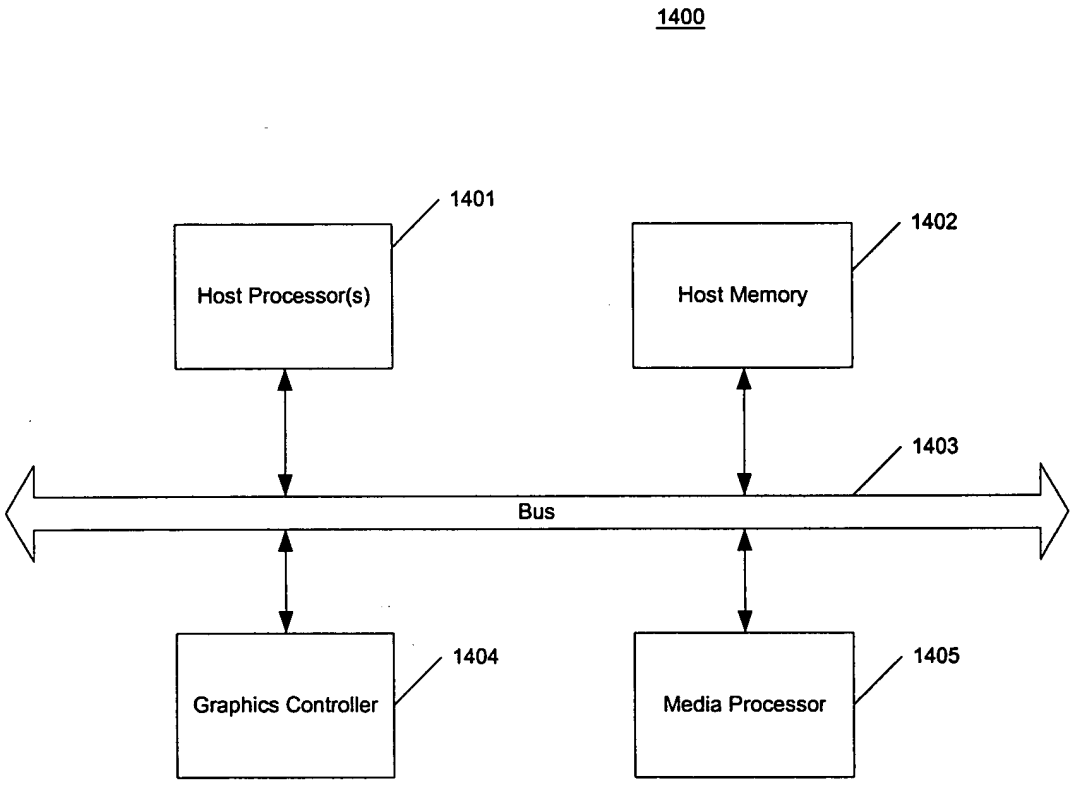

1400
1401
Host Processor(s)
1402
Host Memory
1403
Bus
1404
Graphics Controller
1405
Media Processor

Figure 4A

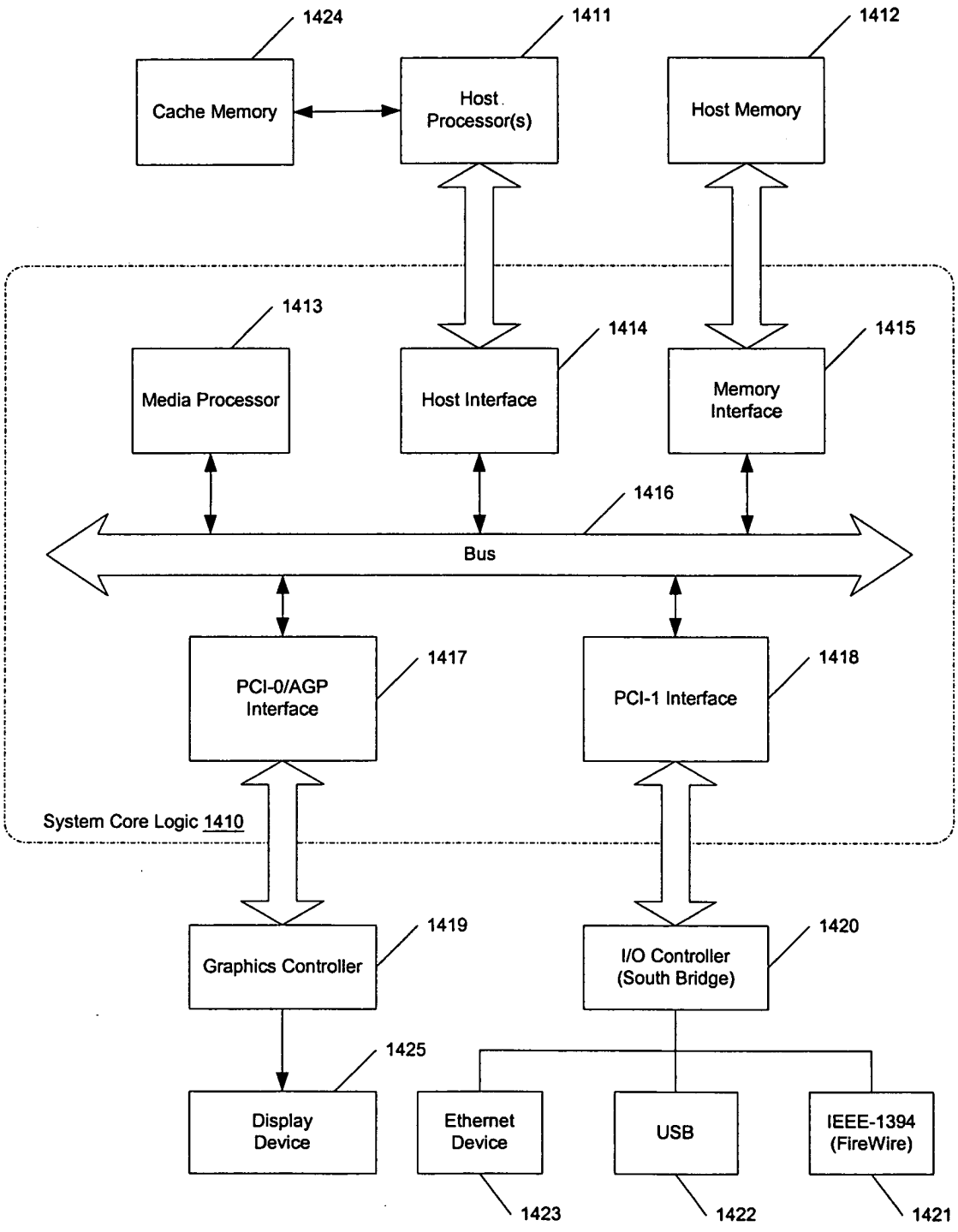

1424
1411
1412
Cache Memory
Host Processor(s)
Host Memory
1413
1414
1415
Media Processor
Host Interface
Memory Interface
1416
Bus
1417
1418
PCI-0/AGP Interface
PCI-1 Interface
System Core Logic 1410
1419
1420
Graphics Controller
I/O Controller (South Bridge)
1425
Display Device
Ethernet Device
USB
IEEE-1394 (FireWire)
1423
1422
1421

Figure 4B

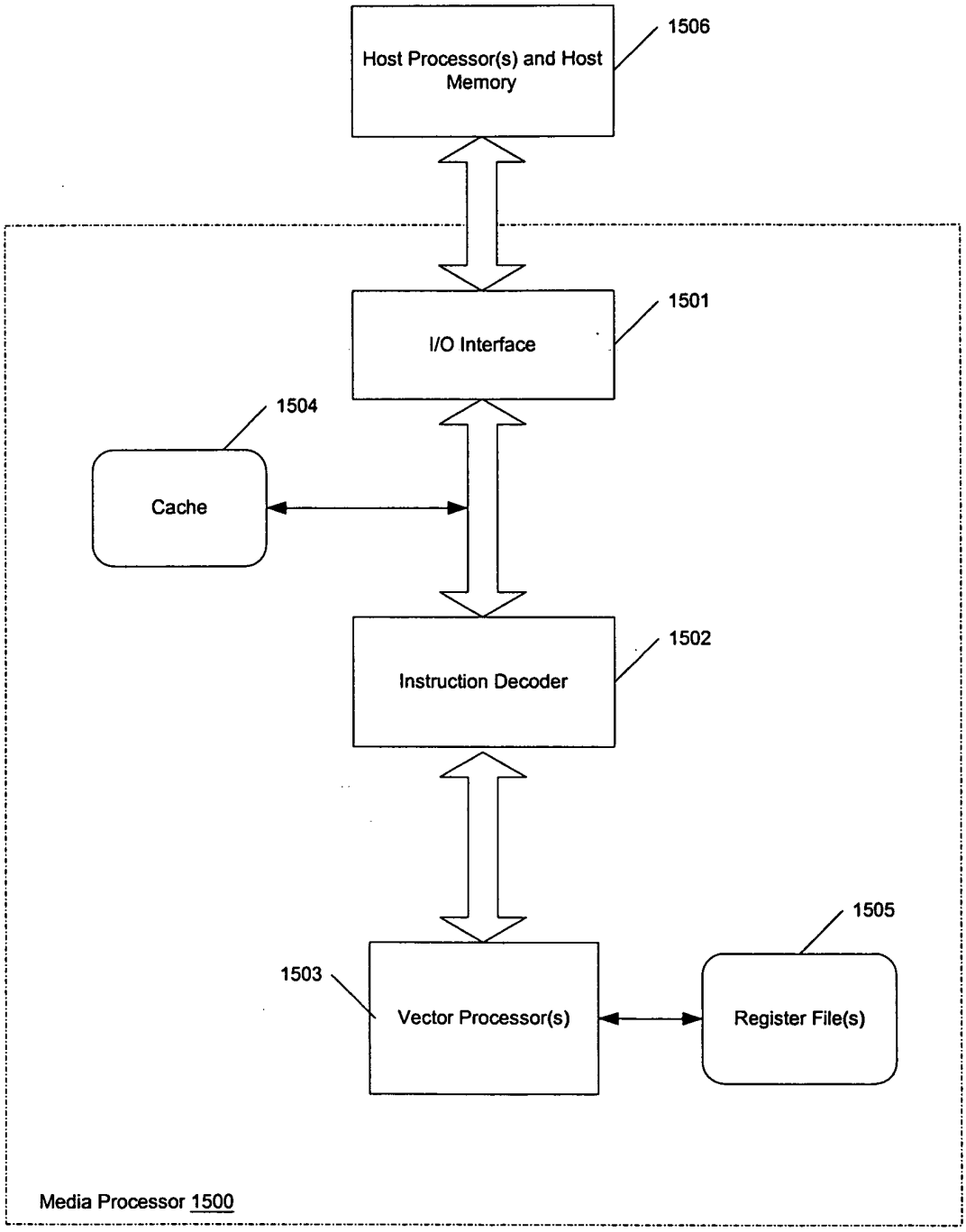

Host Processor(s) and Host Memory
1506
I/O Interface
1501
1504
Cache
Instruction Decoder
1502
1505
1503
Vector Processor(s)
Register File(s)
Media Processor 1500

Figure 5A

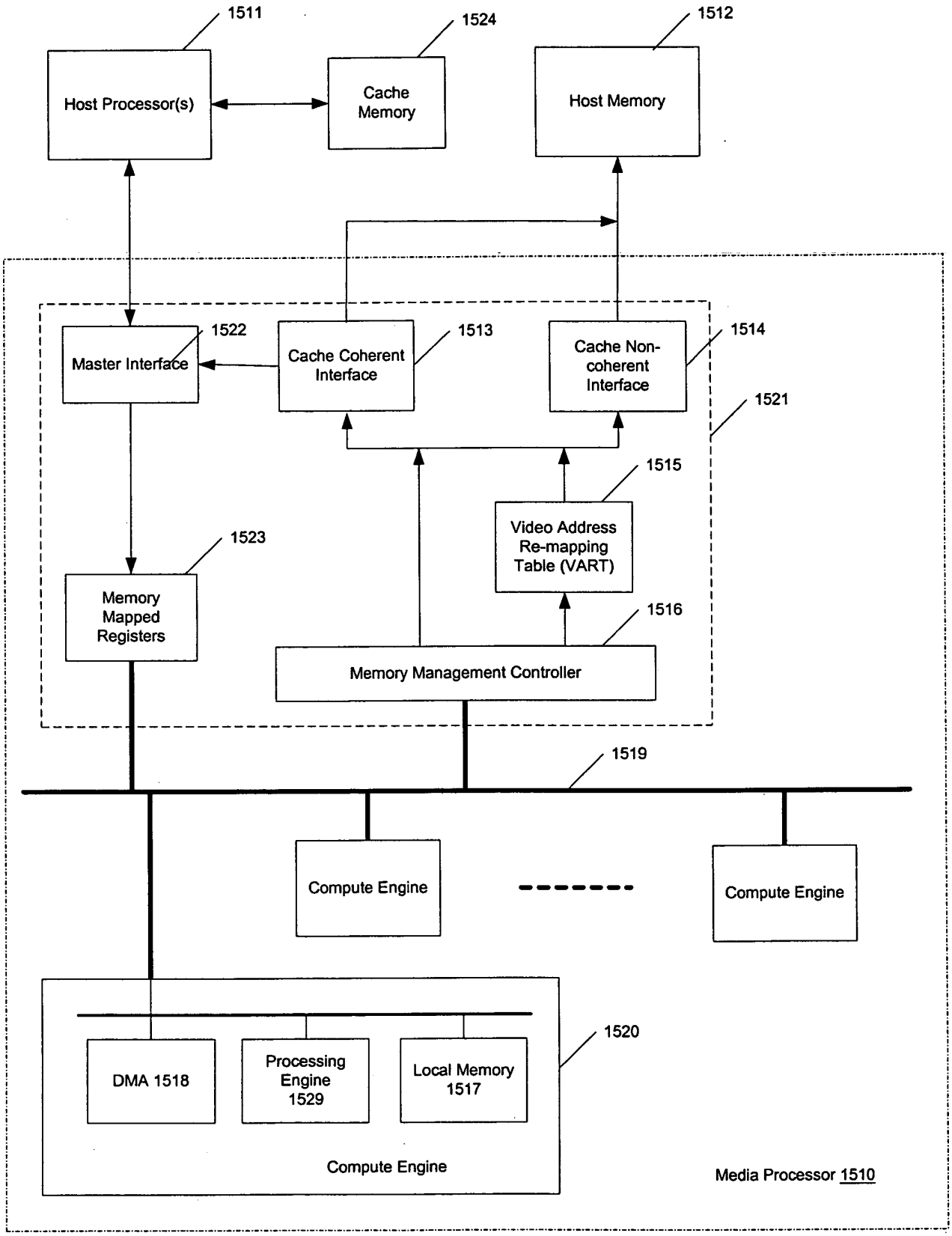

1511
Host Processor(s)
1524
Cache Memory
1512
Host Memory
1522
Master Interface
1513
Cache Coherent Interface
1514
Cache Non-coherent Interface
1521
1515
Video Address Re-mapping Table (VART)
1523
Memory Mapped Registers
1516
Memory Management Controller
1519
Compute Engine
Compute Engine
1520
DMA 1518
Processing Engine 1529
Local Memory 1517
Compute Engine
Media Processor 1510

Figure 5B

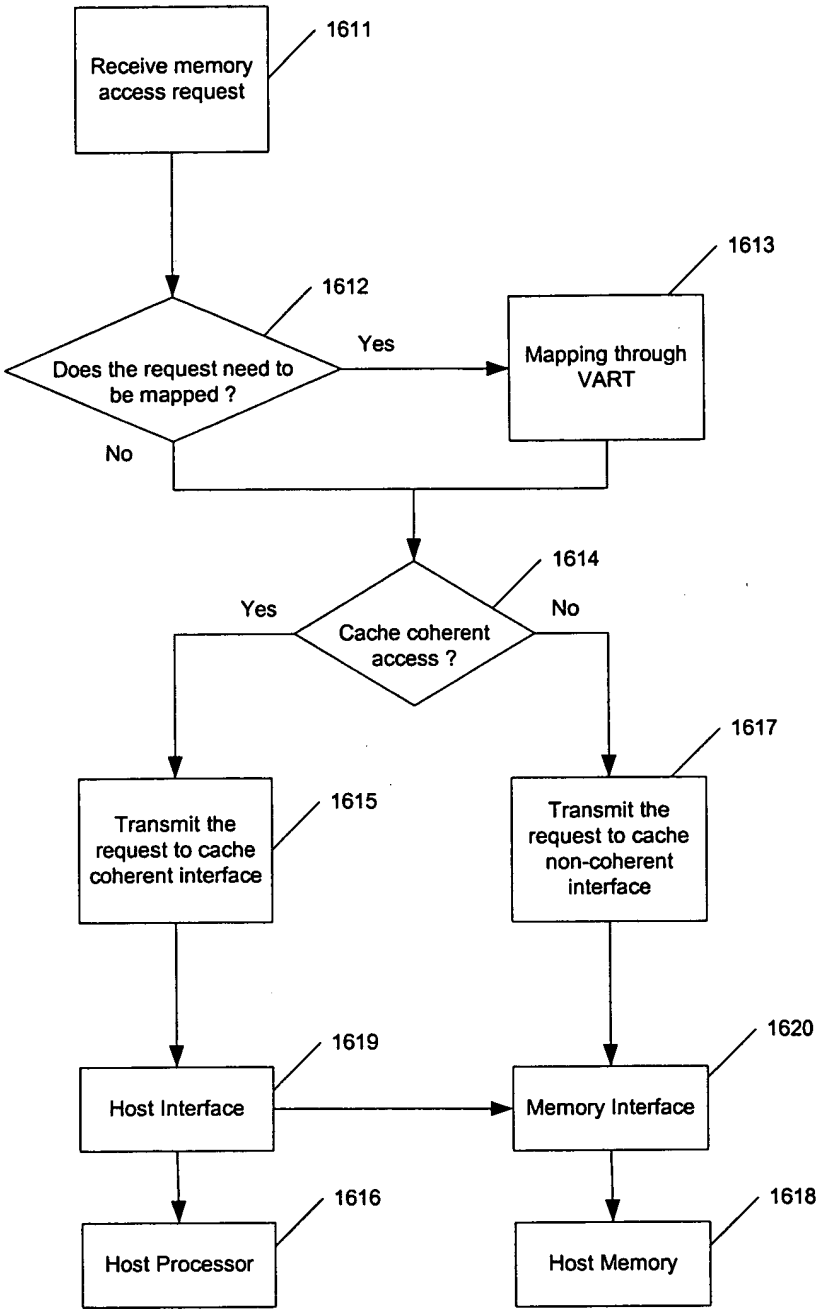

Receive memory access request
1611
Does the request need to be mapped ?
1612
Yes
No
Mapping through VART
1613
Cache coherent access ?
1614
Yes
No
Transmit the request to cache coherent interface
1615
Transmit the request to cache non-coherent interface
1617
Host Interface
1619
Memory Interface
1620
Host Processor
1616
Host Memory
1618

Figure 6

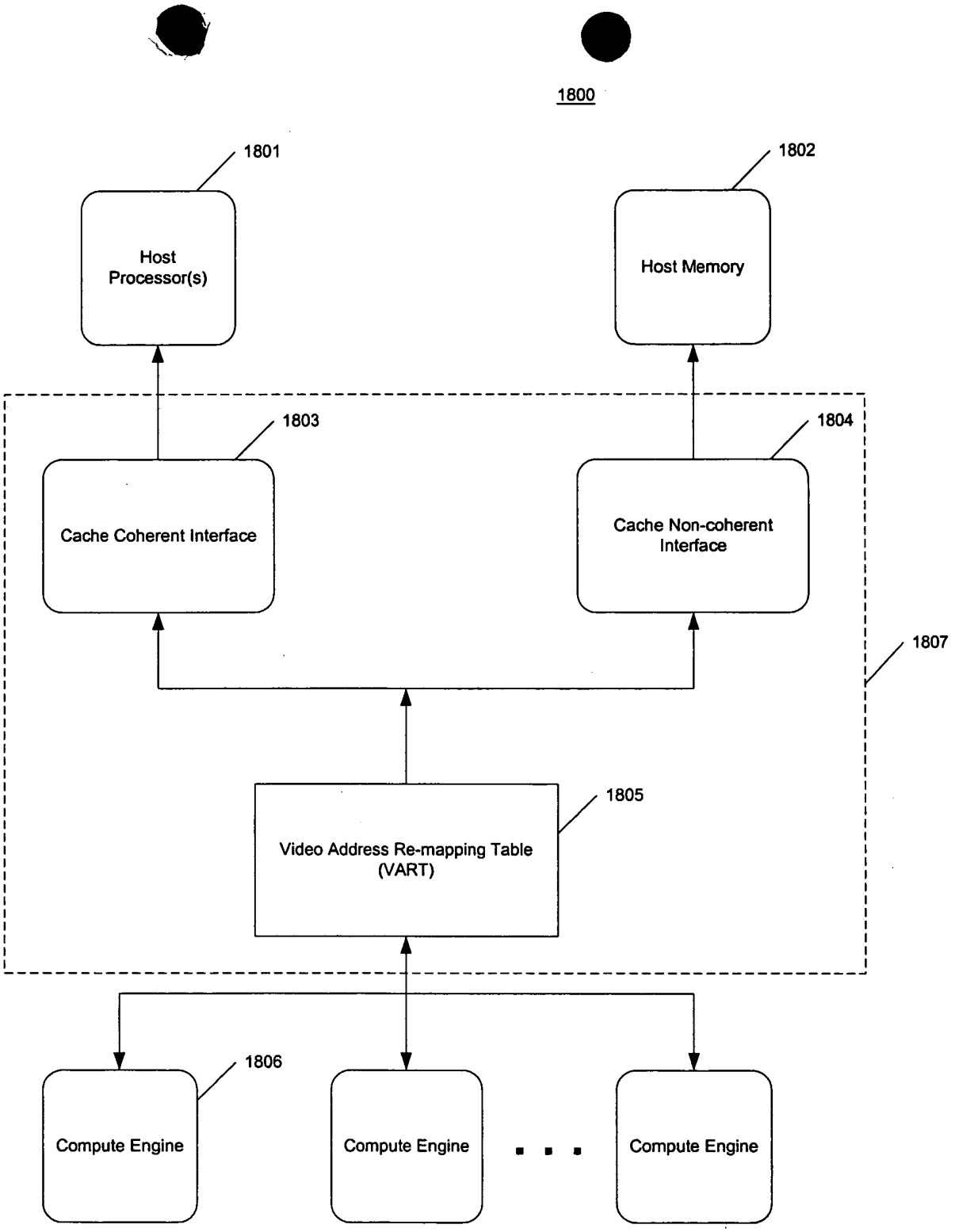
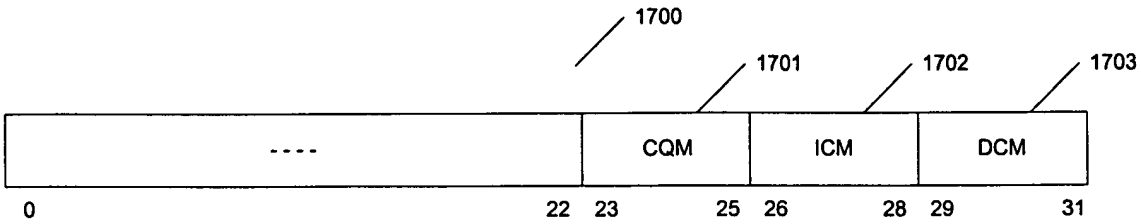

1800
1801
Host Processor(s)
1802
Host Memory
1803
Cache Coherent Interface
1804
Cache Non-coherent Interface
1807
1805
Video Address Re-mapping Table (VART)
1806
Compute Engine
Compute Engine
Compute Engine

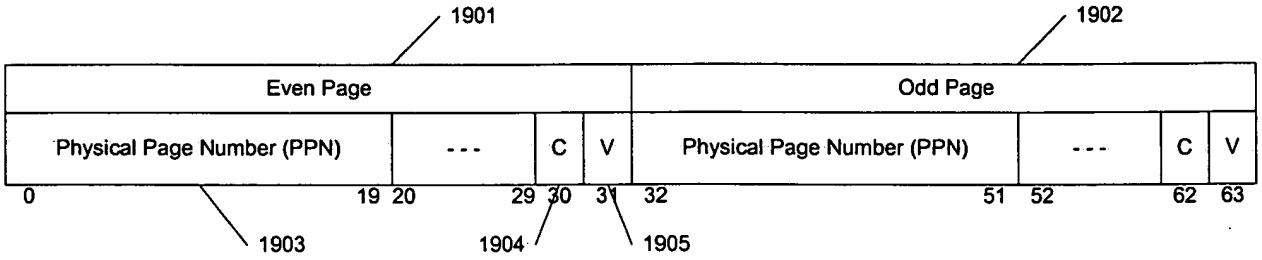
Figure 8



1704

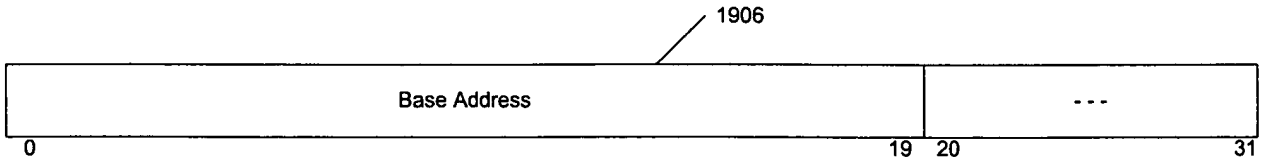
Memory Access Mode Code	
Code	Description
0 - -	Mapped
100	Unmapped and coherent
101	Unmapped and non-coherent
110	if (LogicalAddress[0] = 0) then mapped else unmapped and coherent
111	if (LogicalAddress[0] = 0) then mapped else unmapped and non-coherent

Figure 7

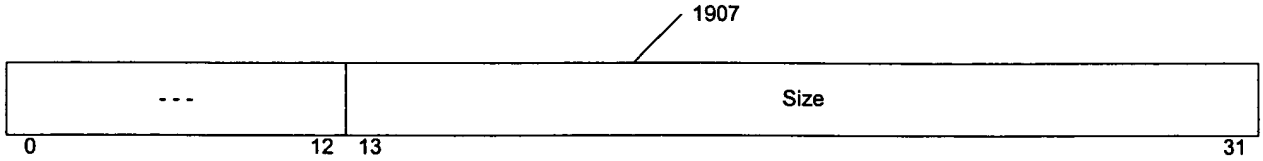

1901
1902
Even Page
Odd Page
Physical Page Number (PPN)

C
V
Physical Page Number (PPN)

C
V
0
19
20
29
30
31
32
51
52
62
63
1903
1904
1905


1906
Base Address

0
19
20
31


1907

Size
0
12
13
31

Figure 9

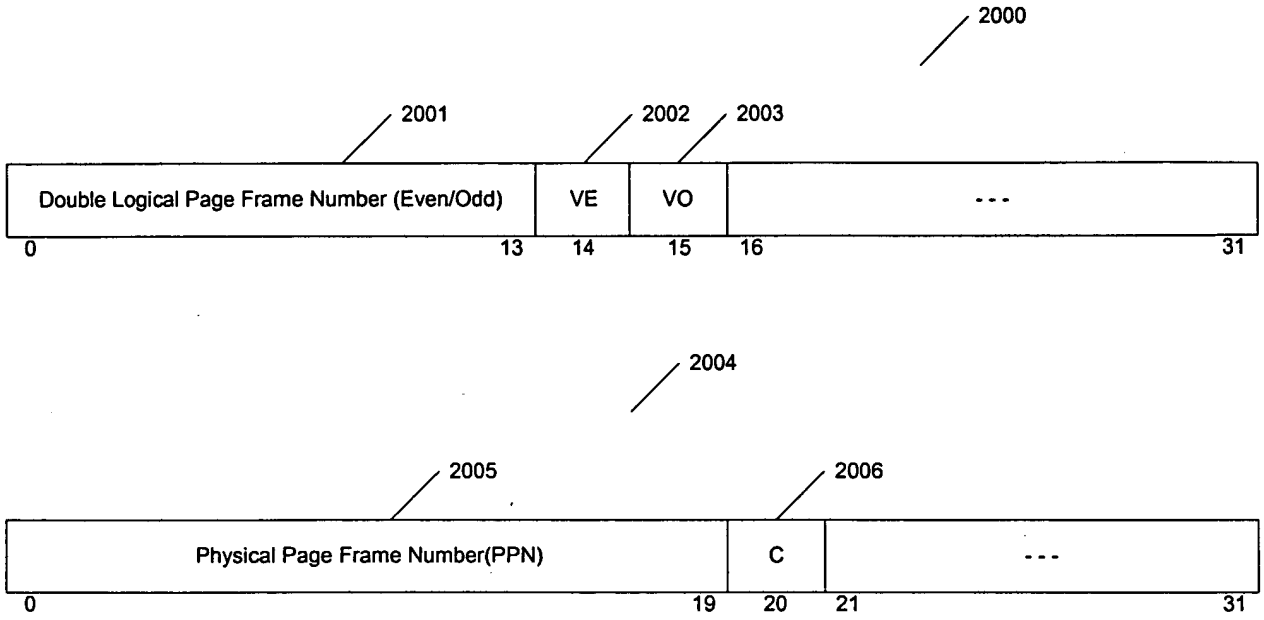


Figure 10A

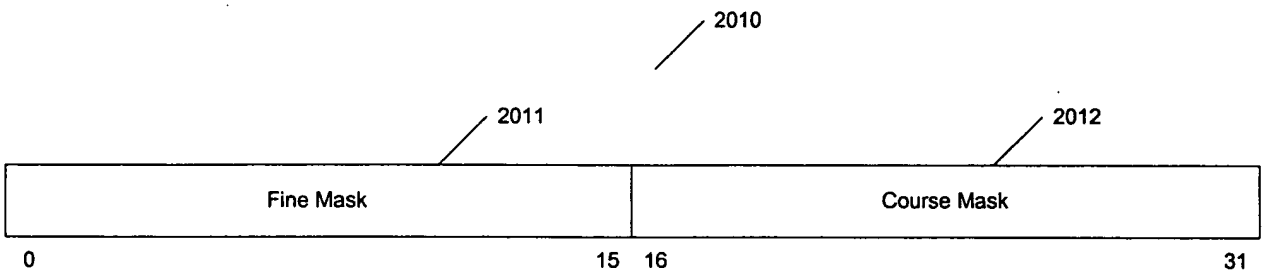


Figure 10B

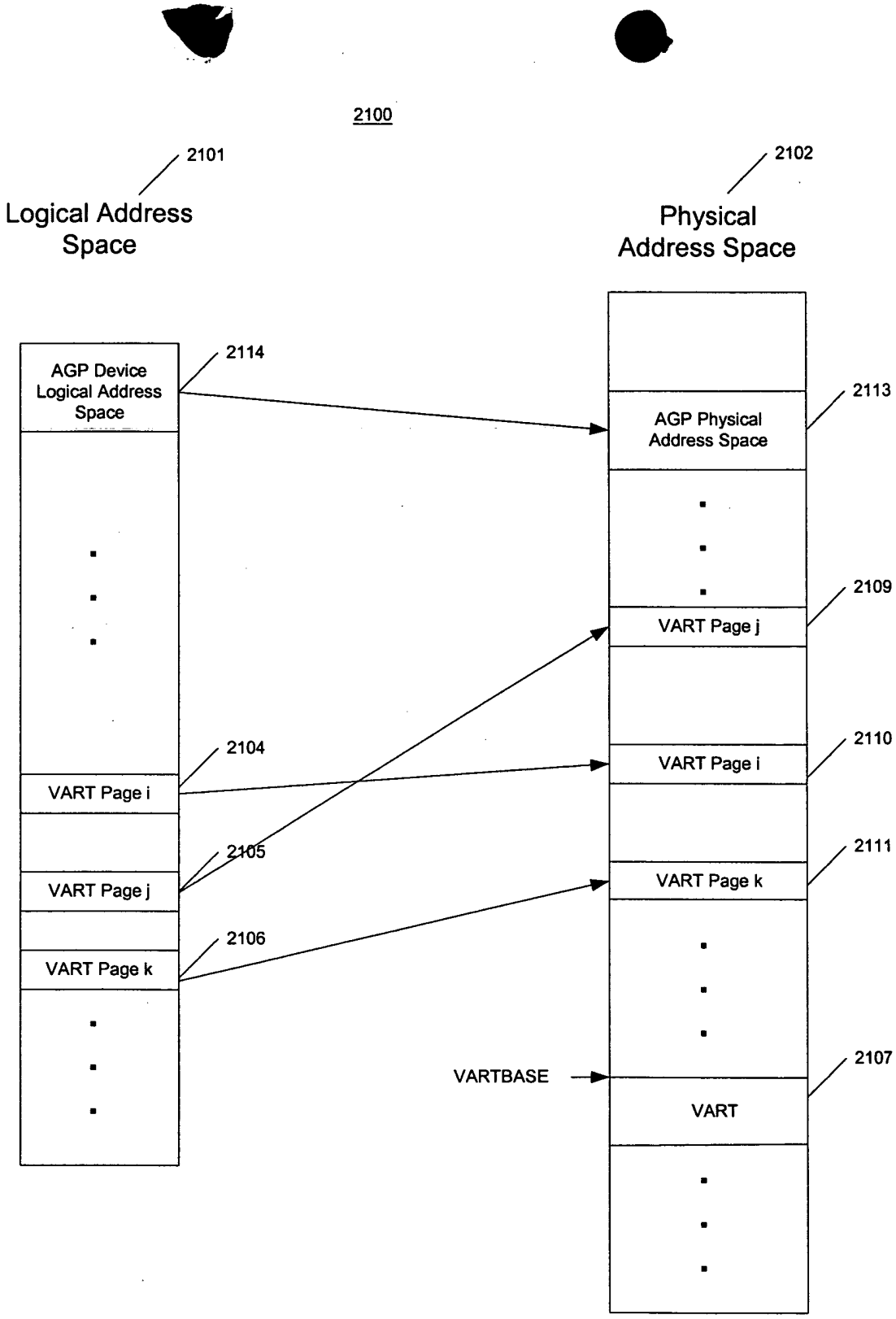

2100
2101
Logical Address Space
2102
Physical Address Space
AGP Device Logical Address Space
2114
2113
AGP Physical Address Space
2109
VART Page j
2104
2110
VART Page i
VART Page i
2105
2111
VART Page j
VART Page k
2106
VART Page k
VARTBASE
2107
VART

Figure 11

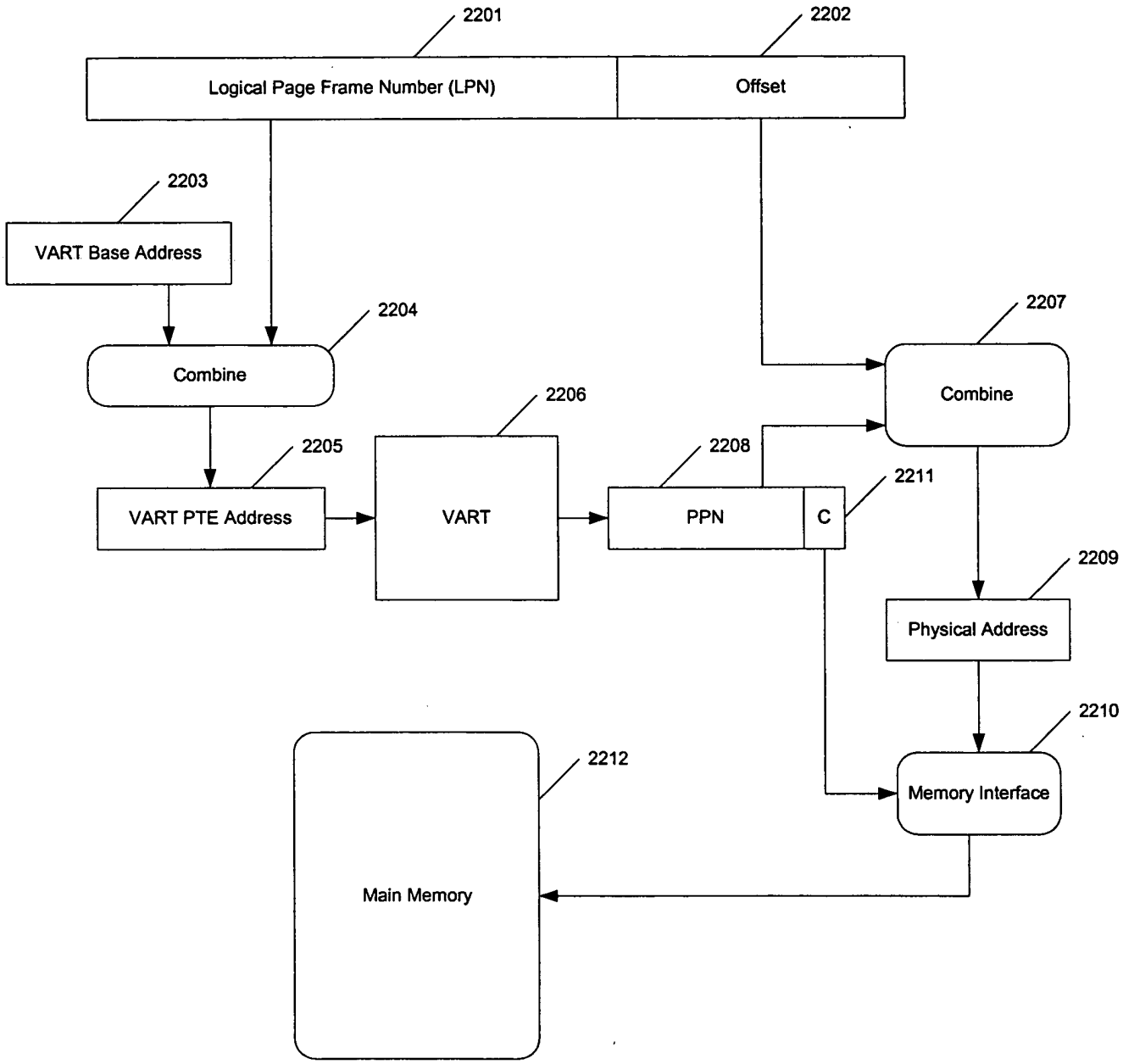

2200
2201
2202
Logical Page Frame Number (LPN)
Offset
2203
VART Base Address
2204
Combine
2207
Combine
2206
2205
2208
2211
VART PTE Address
VART
PPN
C
2209
Physical Address
2210
2212
Memory Interface
Main Memory

Figure 12

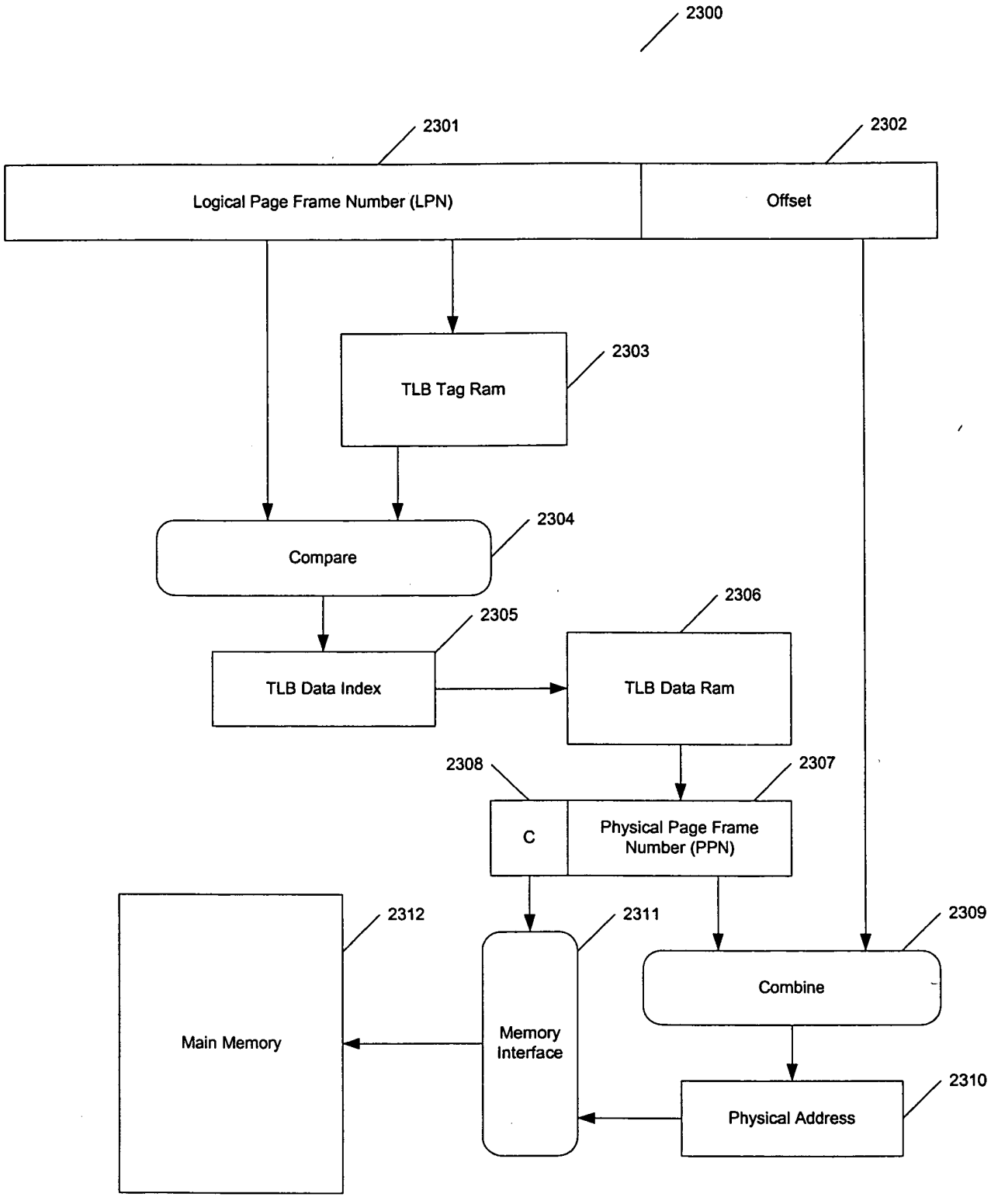

2300
2301
2302
Logical Page Frame Number (LPN)
Offset
TLB Tag Ram
2303
Compare
2304
2306
2305
TLB Data Index
TLB Data Ram
2308
2307
C
Physical Page Frame Number (PPN)
2312
2311
2309
Main Memory
Memory Interface
Combine
2310
Physical Address

Figure 13

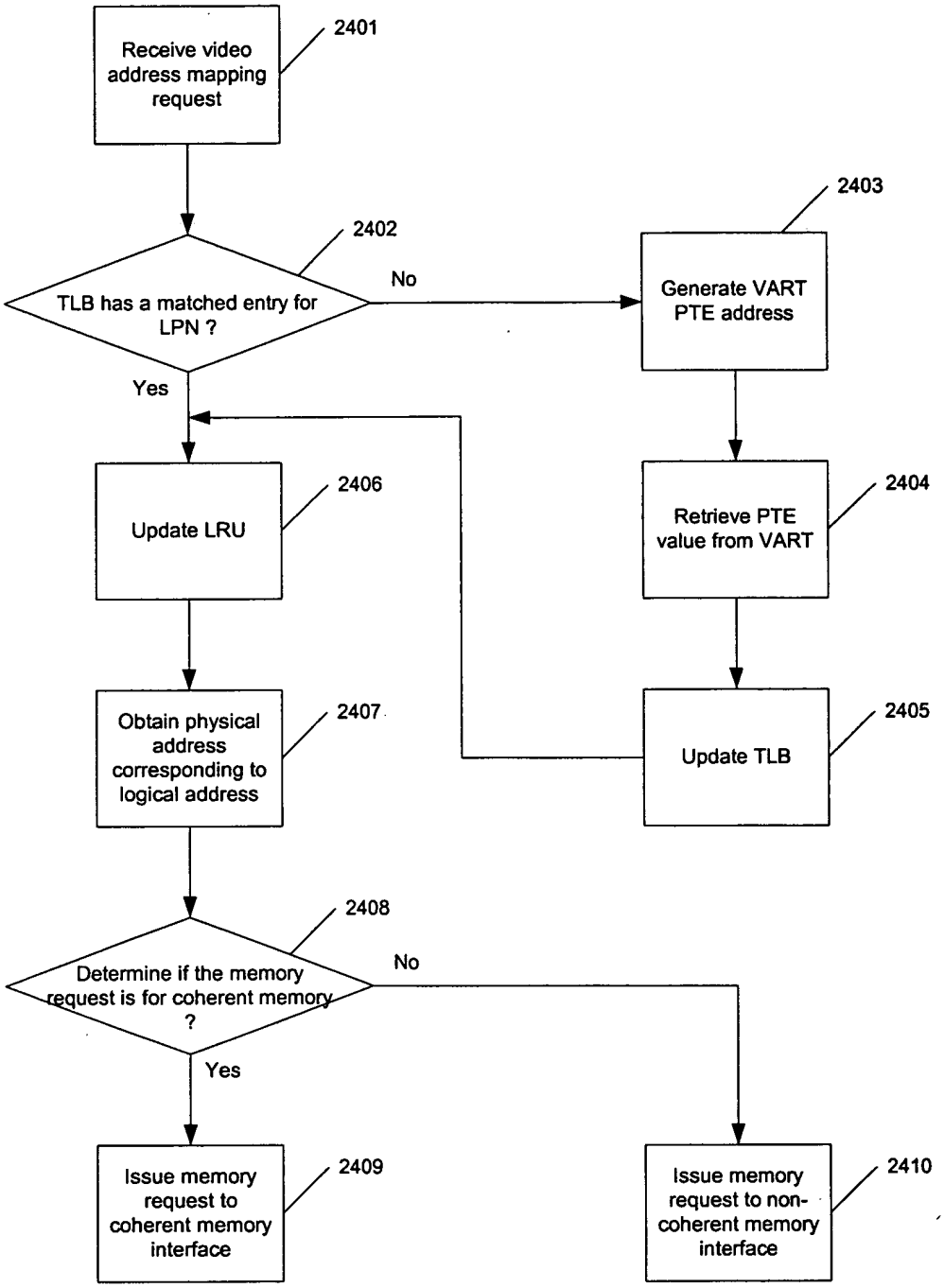

Receive video address mapping request
2401
TLB has a matched entry for LPN ?
2402
No
Yes
Generate VART PTE address
2403
Retrieve PTE value from VART
2404
Update TLB
2405
Update LRU
2406
Obtain physical address corresponding to logical address
2407
Determine if the memory request is for coherent memory ?
2408
No
Yes
Issue memory request to coherent memory interface
2409
Issue memory request to non-coherent memory interface
2410

Figure 14

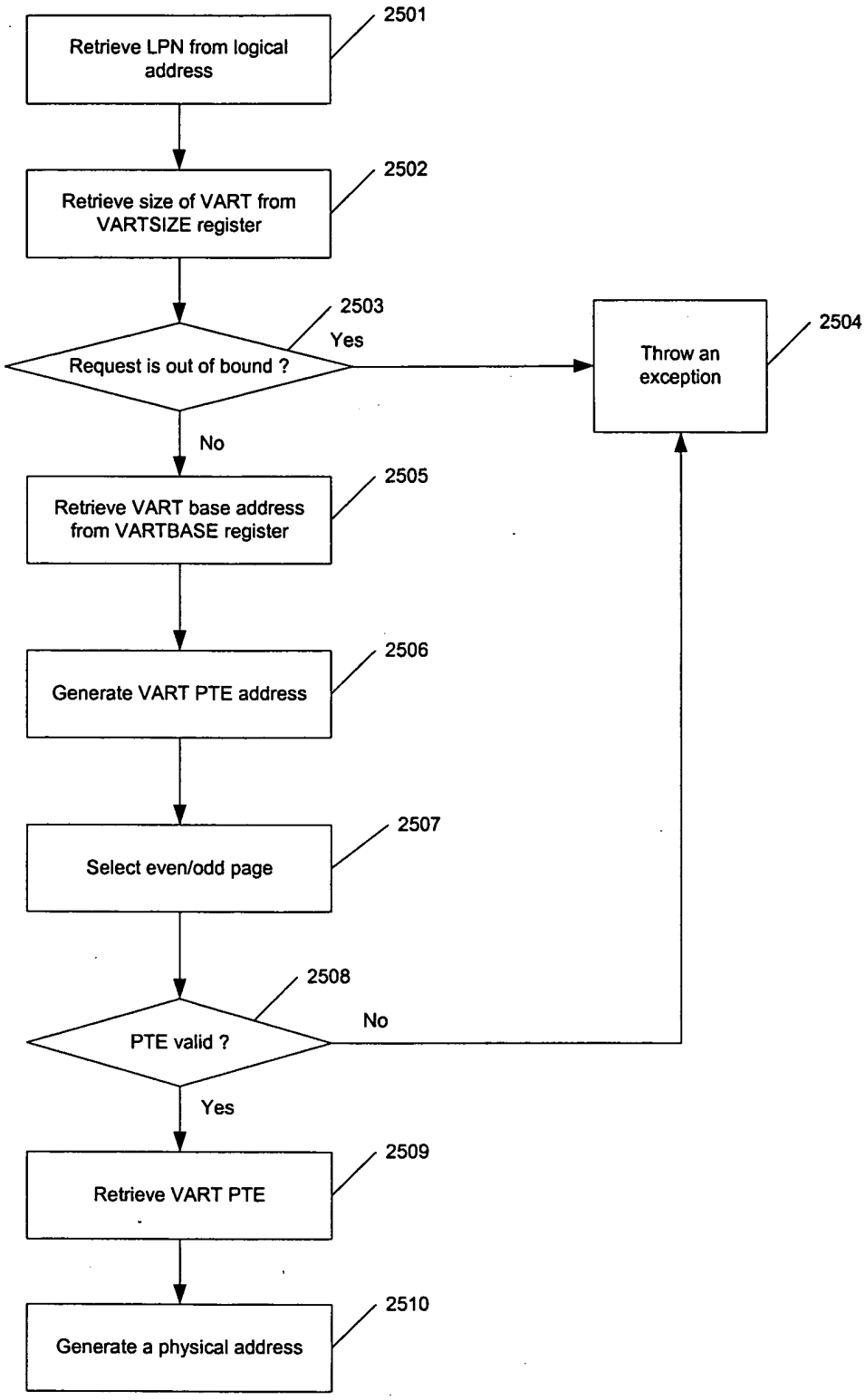

Retrieve LPN from logical address
2501
Retrieve size of VART from VARTSIZE register
2502
Request is out of bound ?
2503
Yes
Throw an exception
2504
No
Retrieve VART base address from VARTBASE register
2505
Generate VART PTE address
2506
Select even/odd page
2507
PTE valid ?
2508
No
Yes
Retrieve VART PTE
2509
Generate a physical address
2510

Figure 15

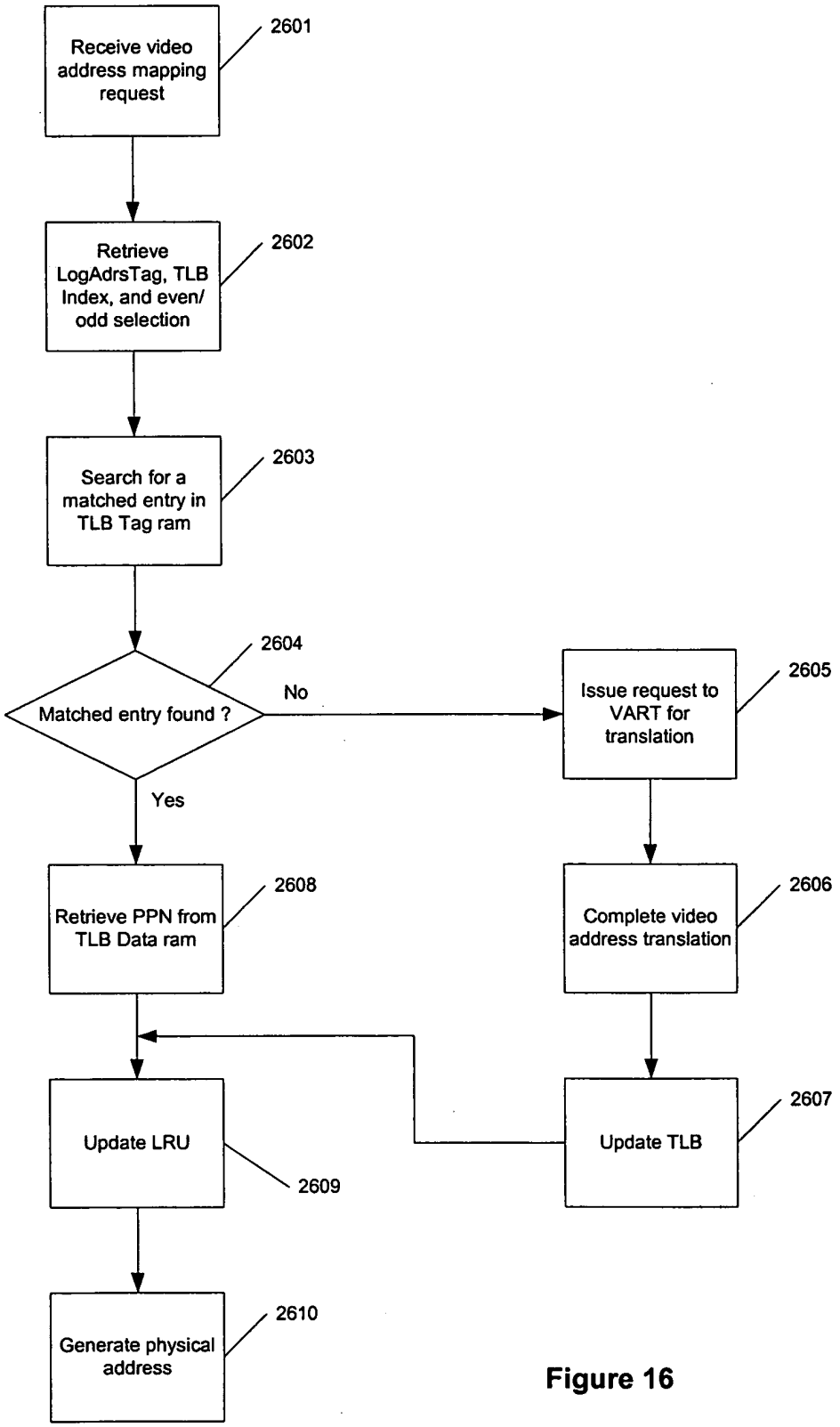

Receive video address mapping request
2601
Retrieve LogAdrsTag, TLB Index, and even/ odd selection
2602
Search for a matched entry in TLB Tag ram
2603
Matched entry found ?
2604
No
Yes
Issue request to VART for translation
2605
Retrieve PPN from TLB Data ram
2608
Complete video address translation
2606
Update LRU
2609
Update TLB
2607
Generate physical address
2610

Figure 16

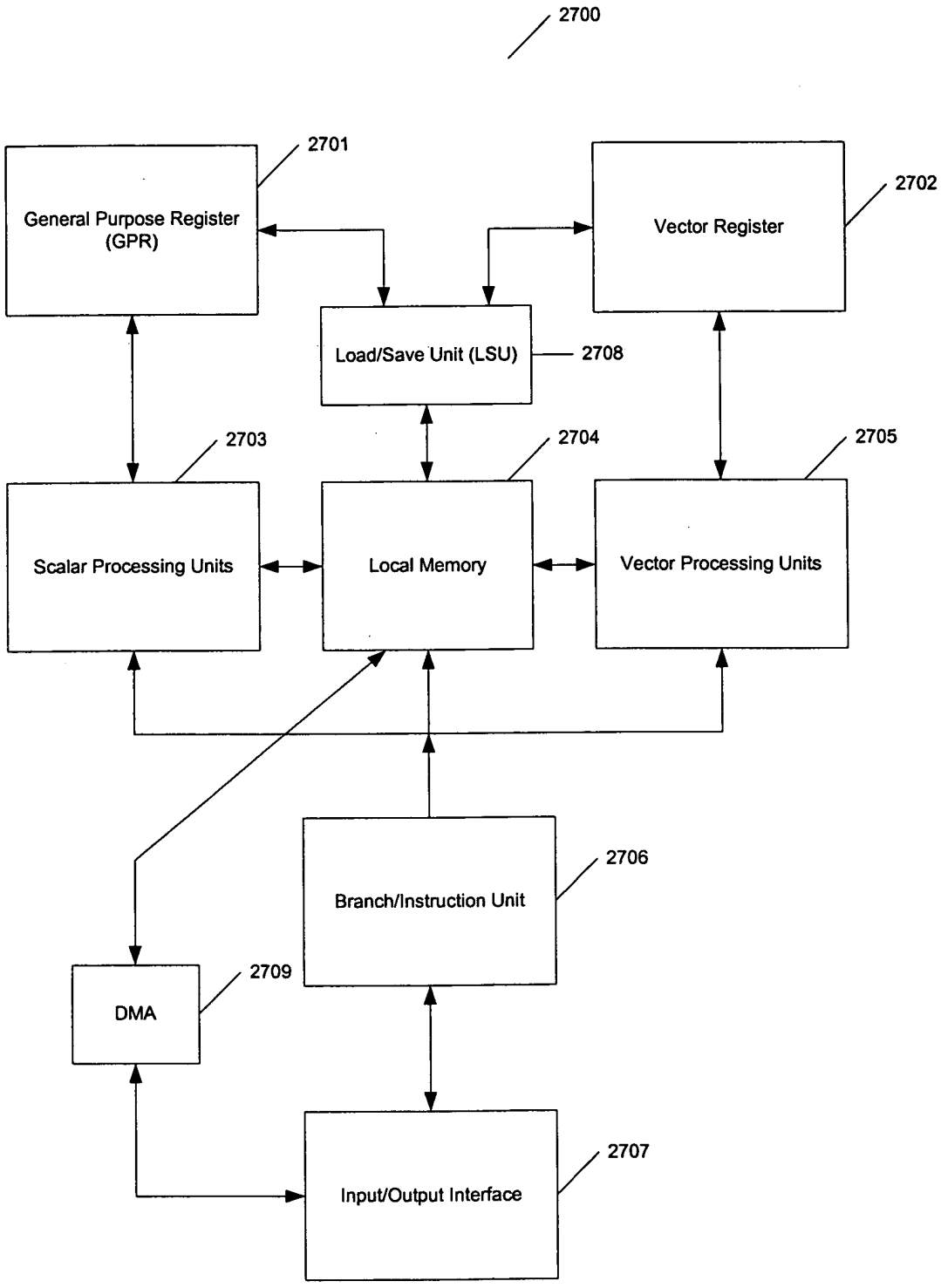

2700
2701
General Purpose Register (GPR)
2702
Vector Register
Load/Save Unit (LSU)
2708
2703
2704
2705
Scalar Processing Units
Local Memory
Vector Processing Units
2706
Branch/Instruction Unit
2709
DMA
2707
Input/Output Interface

Figure 17

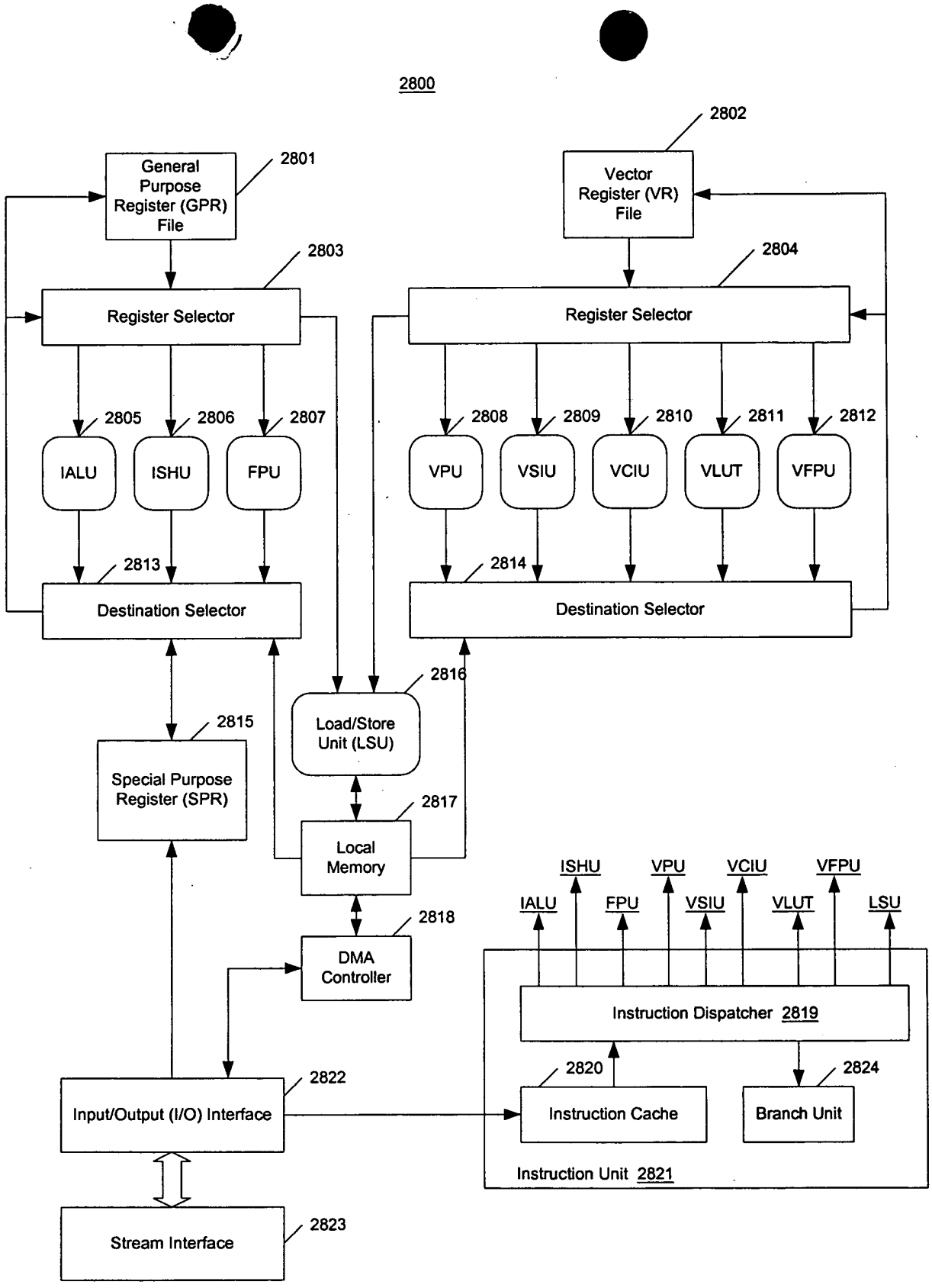

2800
General Purpose Register (GPR) File
2801
Vector Register (VR) File
2802
Register Selector
2803
Register Selector
2804
2805
2806
2807
IALU
ISHU
FPU
2808
2809
2810
2811
2812
VPU
VSIU
VCIU
VLUT
VFPU
2813
Destination Selector
2814
Destination Selector
2816
Load/Store Unit (LSU)
2815
Special Purpose Register (SPR)
2817
Local Memory
2818
DMA Controller
ISHU
VPU
VCIU
VFPU
IALU
FPU
VSIU
VLUT
LSU
Instruction Dispatcher 2819
2820
2824
2822
Input/Output (I/O) Interface
Instruction Cache
Branch Unit
Instruction Unit 2821
2823
Stream Interface

Figure 18

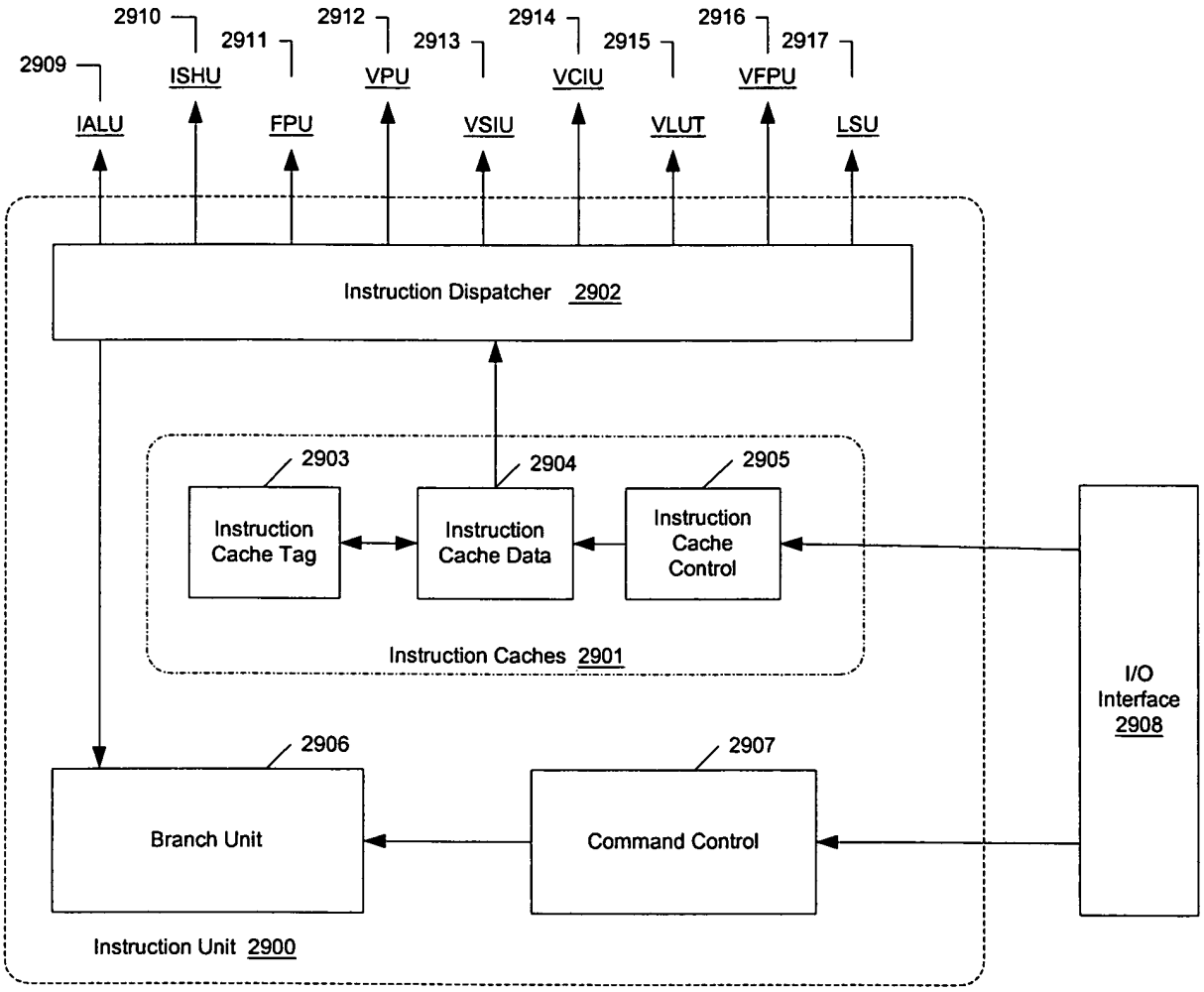

2910
2911
2912
2913
2914
2915
2916
2917
2909
ISHU
VPU
VCIU
VFPU
IALU
FPU
VSIU
VLUT
LSU
Instruction Dispatcher 2902
2903
2904
2905
Instruction Cache Tag
Instruction Cache Data
Instruction Cache Control
Instruction Caches 2901
I/O Interface 2908
2906
2907
Branch Unit
Command Control
Instruction Unit 2900

Figure 19A

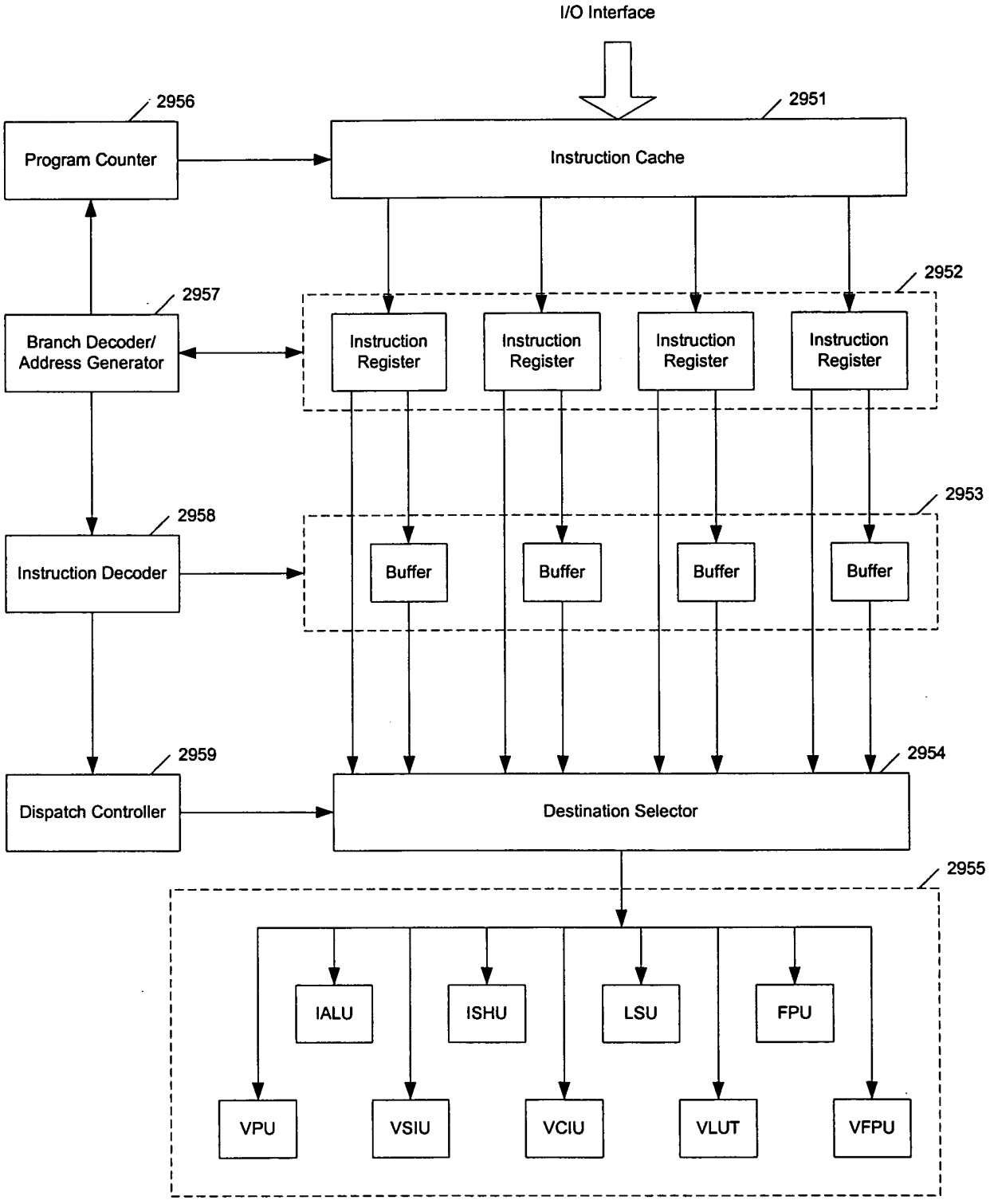

I/O Interface
2956
Program Counter
2951
Instruction Cache
2957
Branch Decoder/
Address Generator
2952
Instruction
Register
Instruction
Register
Instruction
Register
Instruction
Register
2958
Instruction Decoder
2953
Buffer
Buffer
Buffer
Buffer
2959
Dispatch Controller
2954
Destination Selector
2955
IALU
ISHU
LSU
FPU
VPU
VSIU
VCIU
VLUT
VFPU

Figure 19B

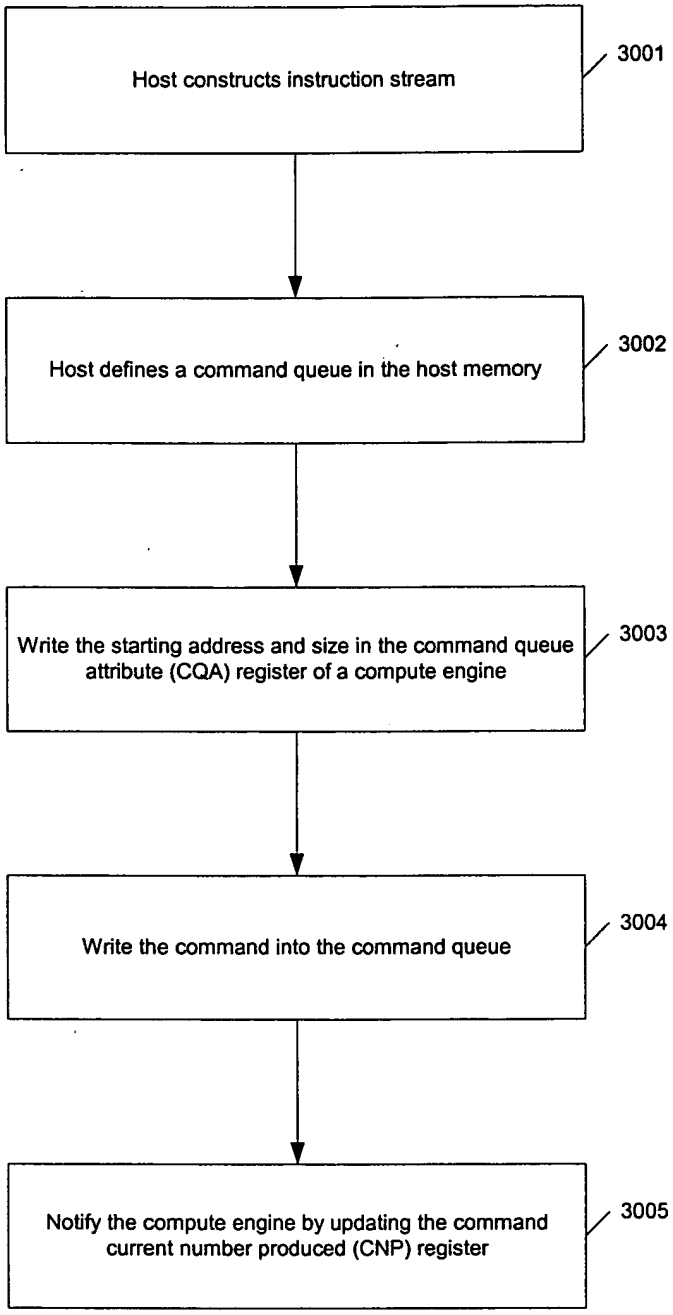

Host constructs instruction stream
3001
Host defines a command queue in the host memory
3002
Write the starting address and size in the command queue attribute (CQA) register of a compute engine
3003
Write the command into the command queue
3004
Notify the compute engine by updating the command current number produced (CNP) register
3005

Figure 20A

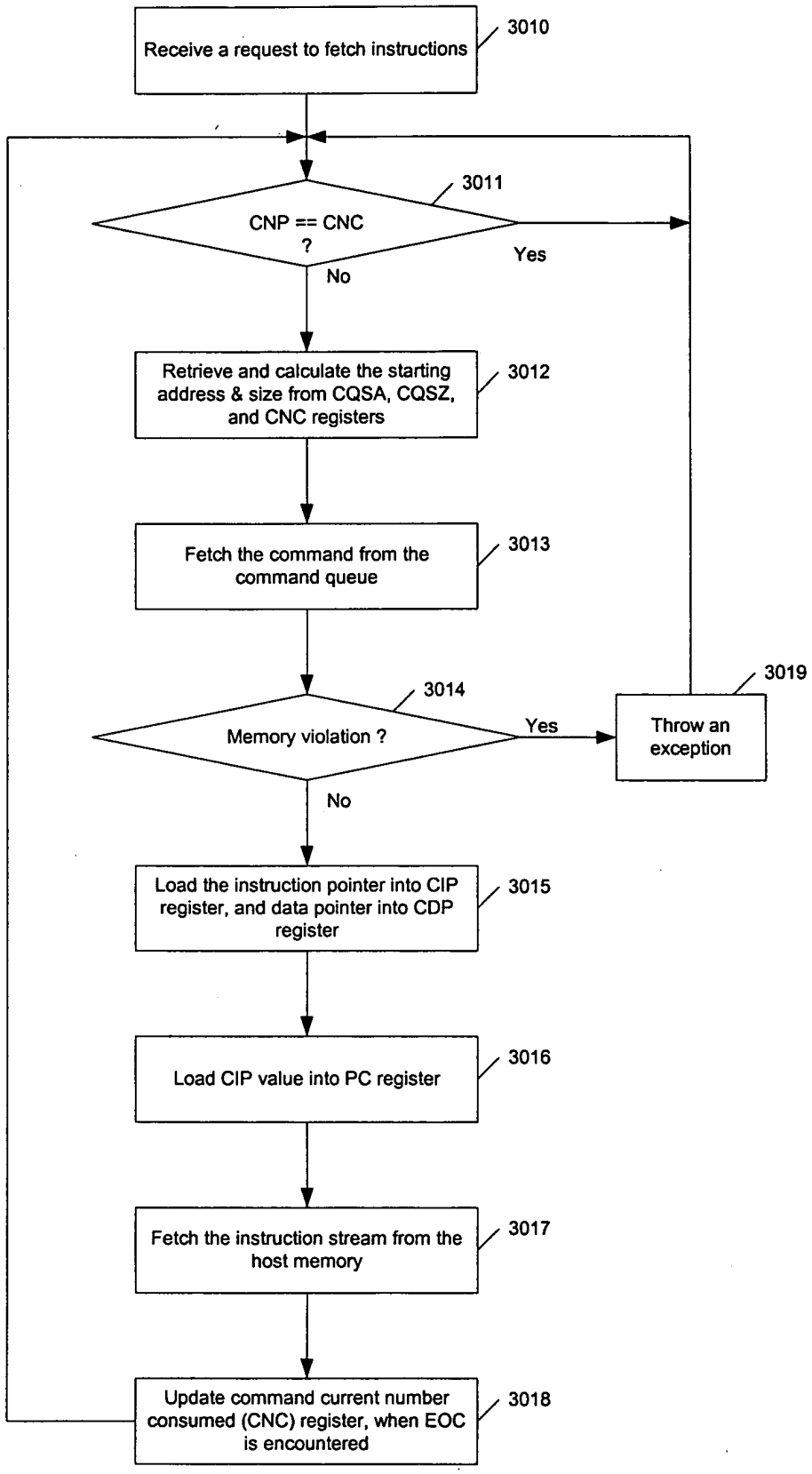

Receive a request to fetch instructions
3010
3011
CNP == CNC ?
Yes
No
Retrieve and calculate the starting address & size from CQSA, CQSZ, and CNC registers
3012
Fetch the command from the command queue
3013
3014
Memory violation ?
Yes
3019
Throw an exception
No
Load the instruction pointer into CIP register, and data pointer into CDP register
3015
Load CIP value into PC register
3016
Fetch the instruction stream from the host memory
3017
Update command current number consumed (CNC) register, when EOC is encountered
3018

Figure 20B

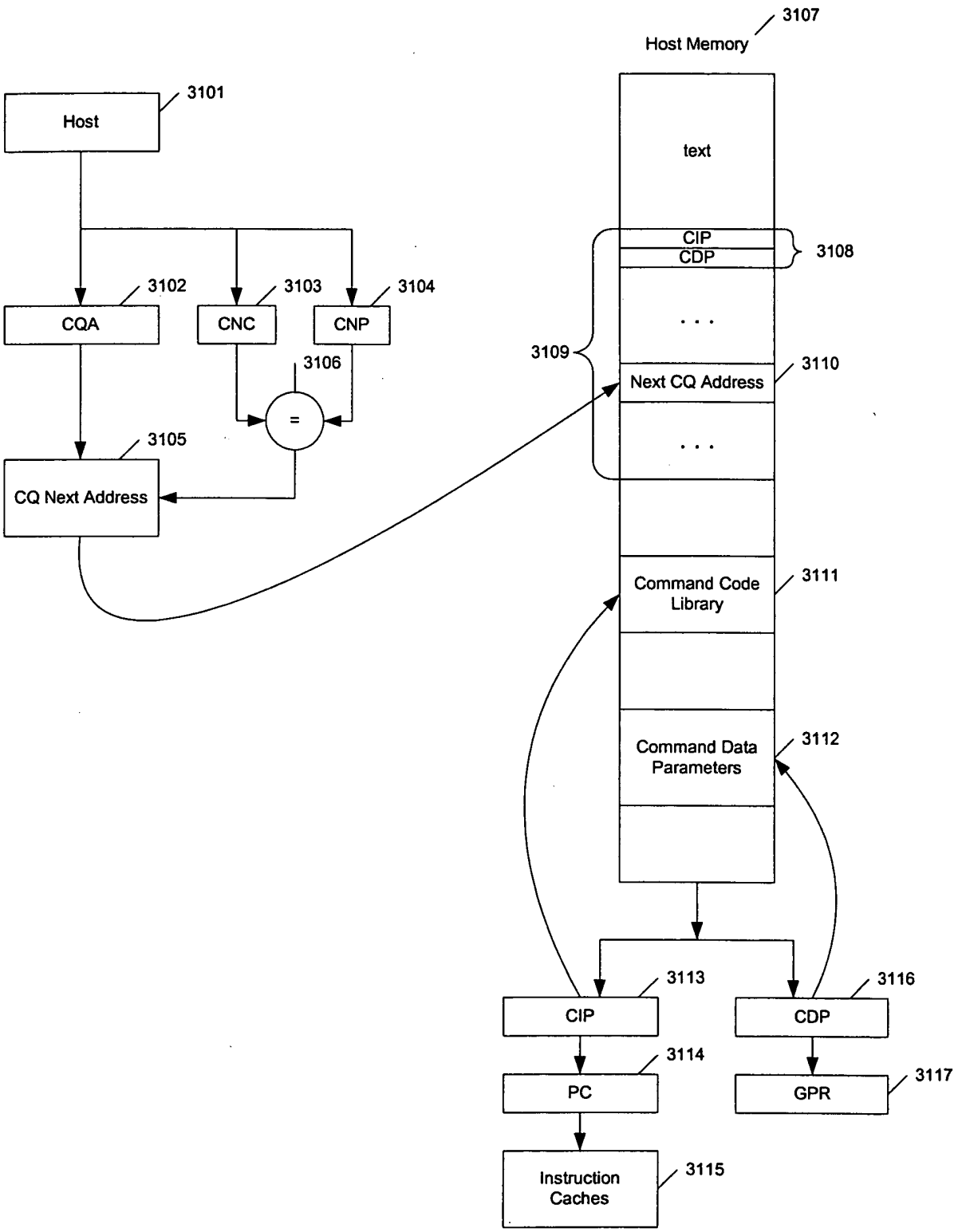
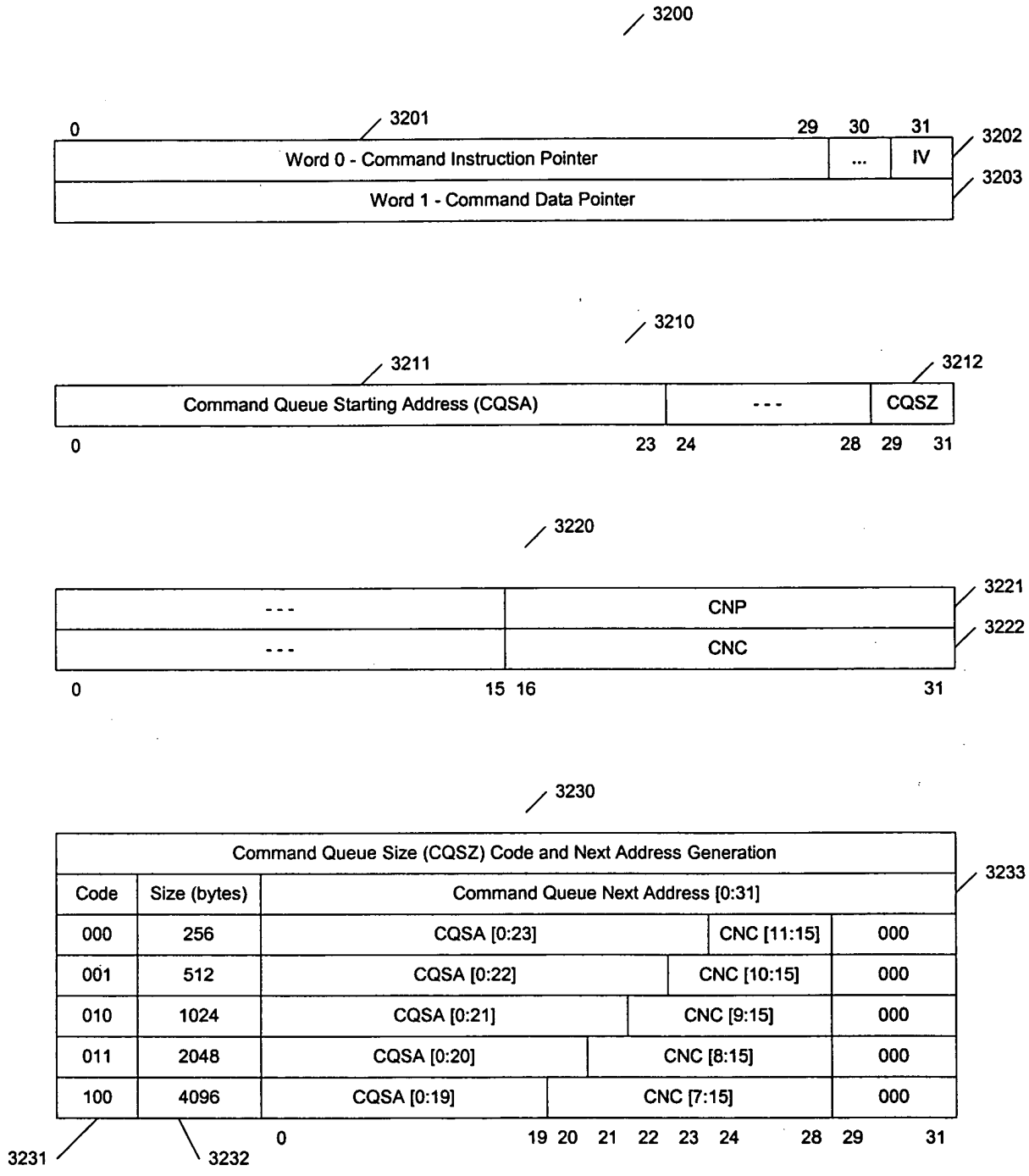

3107
Host Memory
3101
Host
text
CIP
CDP
3108
3102
3103
3104
CQA
CNC
CNP
3106
3109
3110
Next CQ Address
=
3105
CQ Next Address
Command Code Library
3111
Command Data Parameters
3112
3113
3116
CIP
CDP
3114
3117
PC
GPR
Instruction Caches
3115

Figure 21


3200
3201
0
29
30
31
3202
Word 0 - Command Instruction Pointer
...
IV
3203
Word 1 - Command Data Pointer
3210
3211
3212
Command Queue Starting Address (CQSA)

CQSZ
0
23
24
28
29
31
3220
3221

CNP
3222

CNC
0
15
16
31
3230
Command Queue Size (CQSZ) Code and Next Address Generation
3233
Code
Size (bytes)
Command Queue Next Address [0:31]
000
256
CQSA [0:23]
CNC [11:15]
000
001
512
CQSA [0:22]
CNC [10:15]
000
010
1024
CQSA [0:21]
CNC [9:15]
000
011
2048
CQSA [0:20]
CNC [8:15]
000
100
4096
CQSA [0:19]
CNC [7:15]
000
0
19
20
21
22
23
24
28
29
31
3231
3232

Figure 22

3300

Priority Number	Functional Group Name
0	IALU - Integer Arithmetic/Logical Unit
1	ISHU - Integer Shift Unit
2	LSU - Load/Store Unit
3	VPU - Vector Permute Unit
4	VSIU - Vector Simple Integer Unit
5	VCIU - Vector Complex Integer Unit
6	VLUT - Vector Look-up Table Unit
7	BRU - Branch Unit

3301

3302

Figure 23

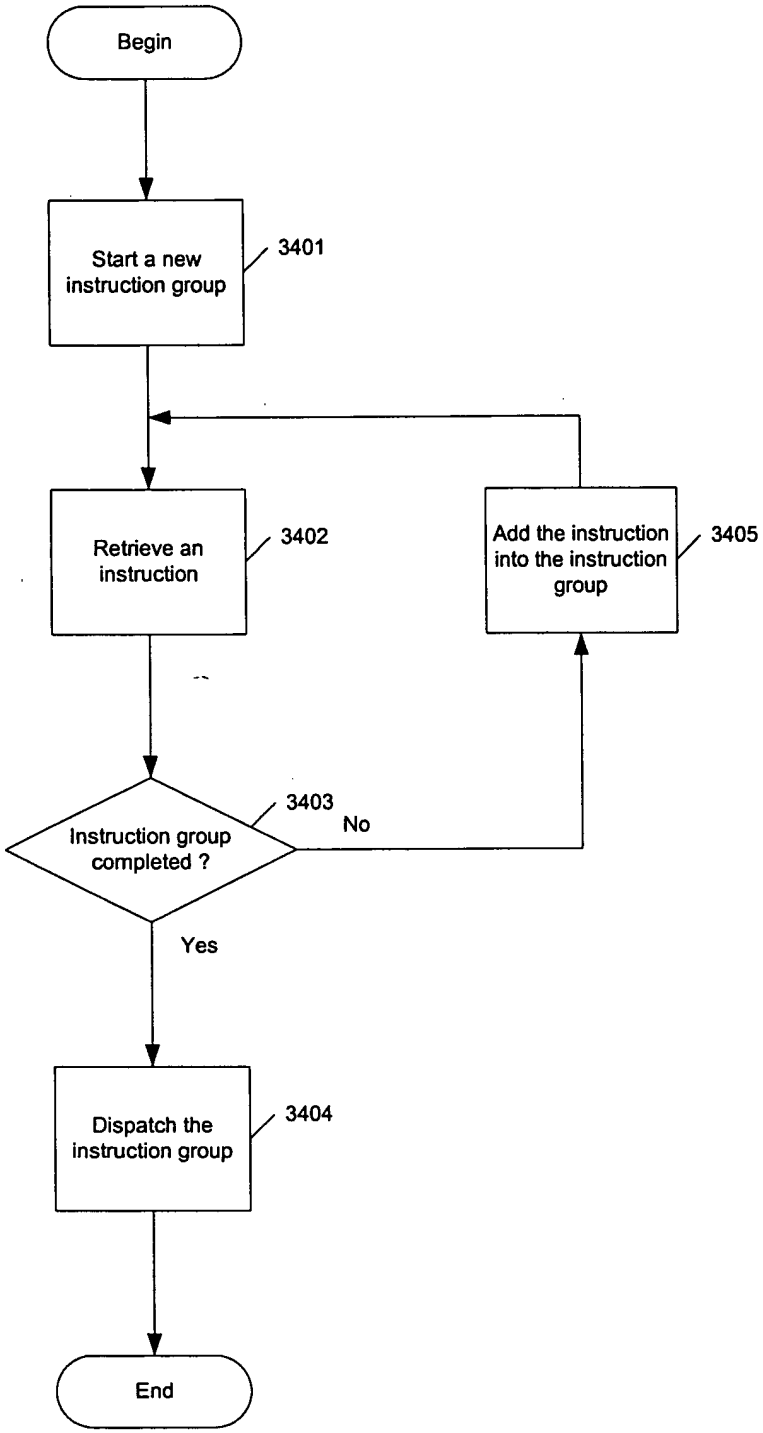

Begin
Start a new instruction group
3401
Retrieve an instruction
3402
Add the instruction into the instruction group
3405
Instruction group completed ?
3403
No
Yes
Dispatch the instruction group
3404
End

Figure 24

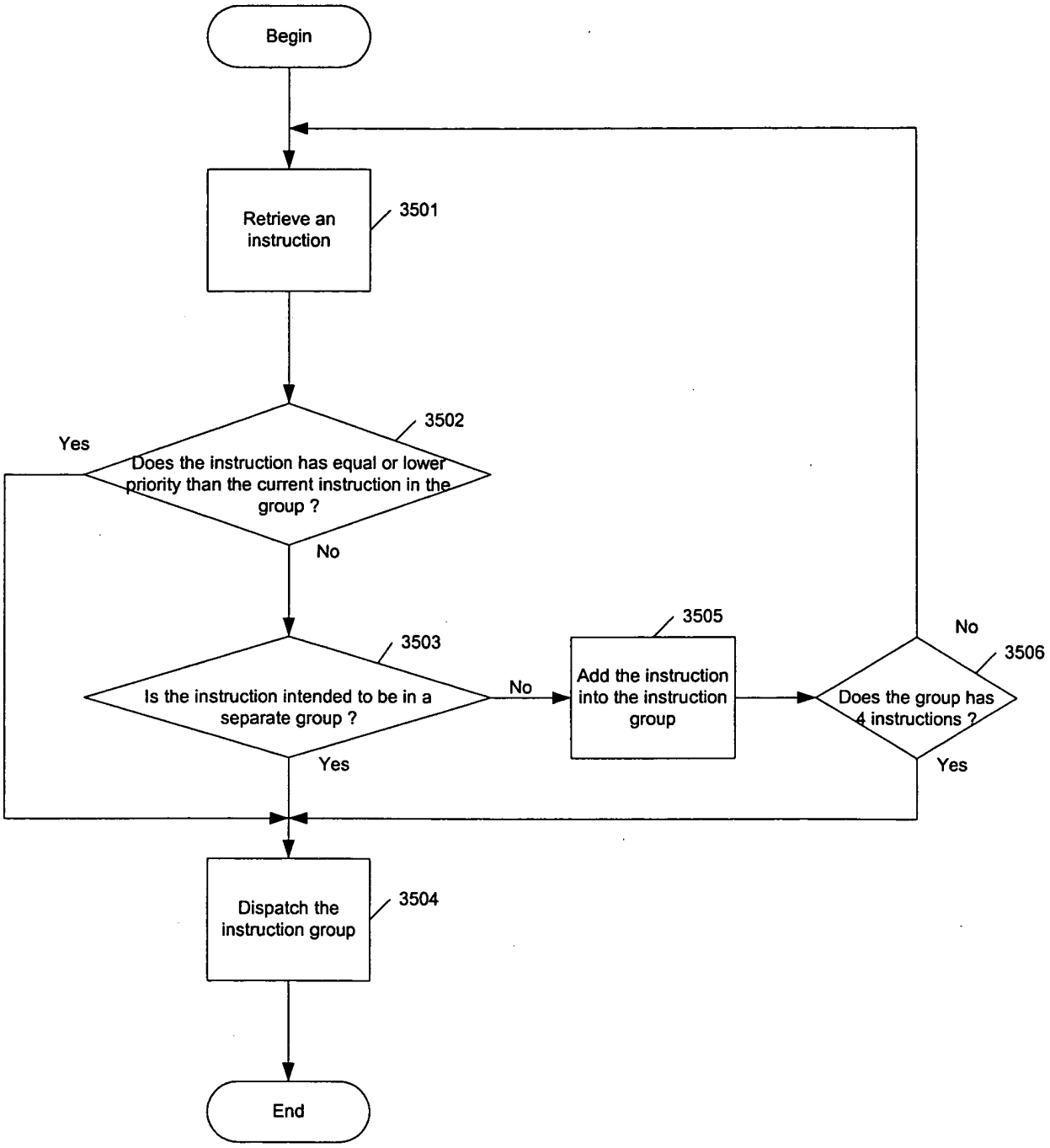

Begin
Retrieve an instruction
3501
3502
Yes
Does the instruction has equal or lower priority than the current instruction in the group ?
No
3503
Is the instruction intended to be in a separate group ?
No
Yes
3505
Add the instruction into the instruction group
No
3506
Does the group has 4 instructions ?
Yes
Dispatch the instruction group
3504
End

Figure 25

3600

Functional Unit	Latency	Dispatch Rate
IALU - not multipy or divide	2	1
IALU - multiply	19	19
IALU - divide	35	35
ISHU	2	1
LSU - non-DMA address update	2	1
LSU - non-DMA load data update	3	1
LSU - non-DMA store	1	1
LSU - DMA instructions	1	1
VPU	2	1
VSIU	2	1
VCIU	6	1
VLUT - reads, vvld	4	1
VLUT - writes	1	1
Branch instruction	1	1

Figure 26

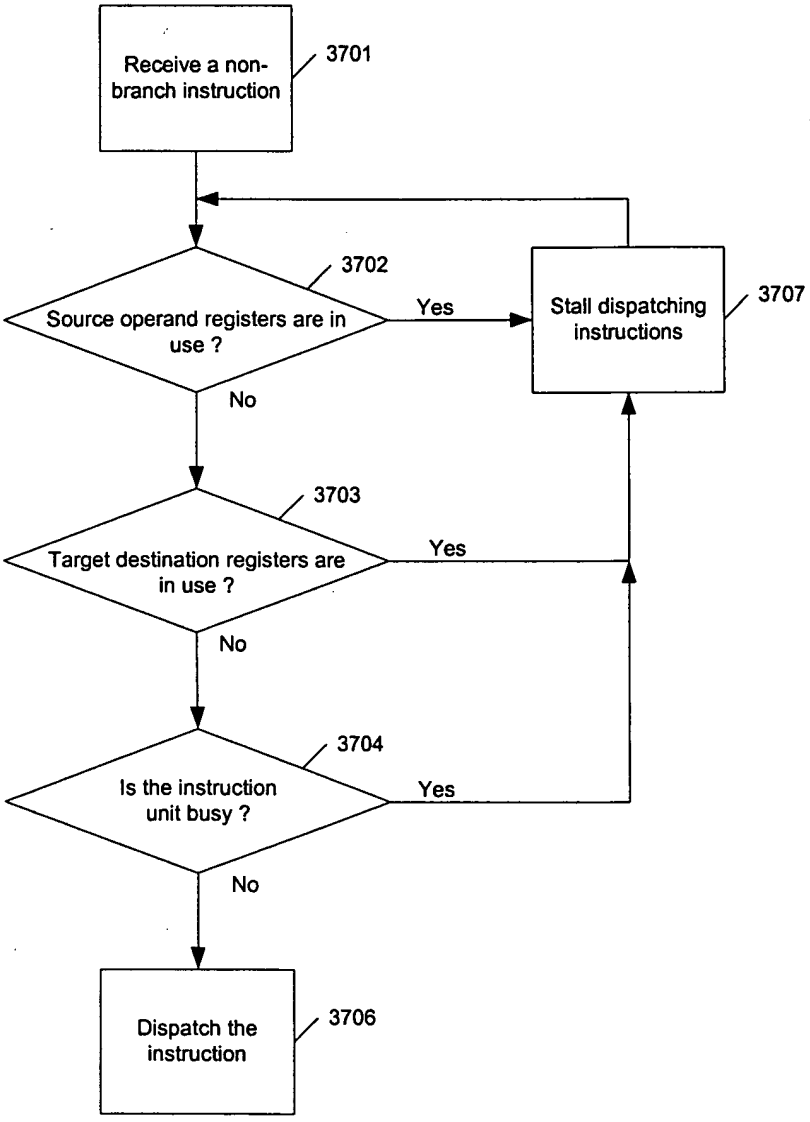

Receive a non-branch instruction
3701
3702
Source operand registers are in use ?
Yes
Stall dispatching instructions
3707
No
3703
Target destination registers are in use ?
Yes
No
3704
Is the instruction unit busy ?
Yes
No
Dispatch the instruction
3706

Figure 27

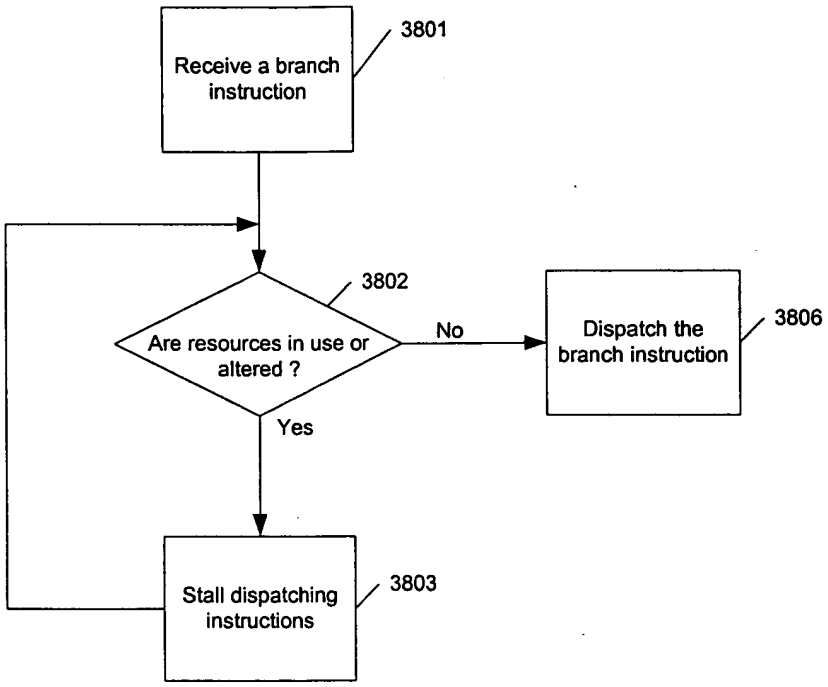
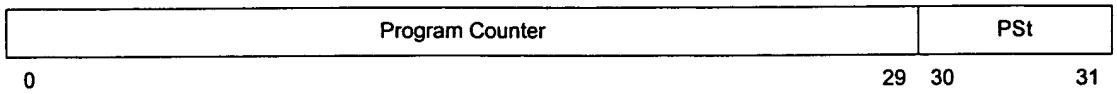

Receive a branch instruction
3801
3802
Are resources in use or altered ?
No
Dispatch the branch instruction
3806
Yes
Stall dispatching instructions
3803

Figure 28

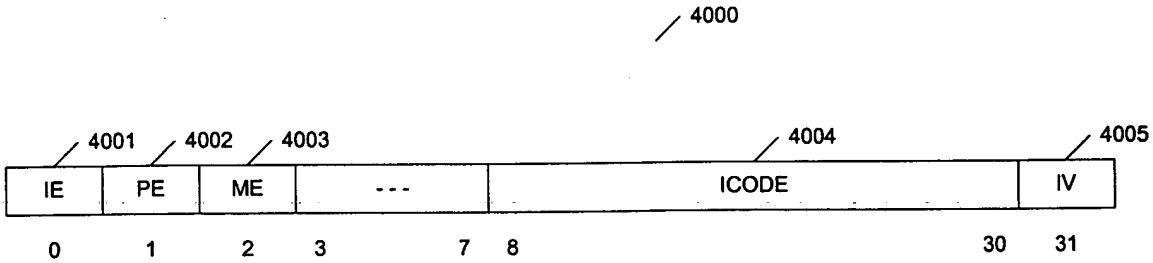
3900



3901

Pst	Name	Description
00	Idle	CQ counters are equal and no current command executing. Program counter is invalid.
01	Run	Command was executing. Program counter points to next instruction that would have been executed.
10	IWait	Command was executing, but instruction fetching has stopped due to a previous exception. Program counter points to the next instruction that would have been executed.
11	CWait	Command was not executing due to an exception in fetching the command. Program counter is invalid.

Figure 29



4006

Name	Descriptions
IE	Illegal Opcode Exception. Occurs whenever an illegal Opcode is fetched for execution. Cleared when read by the host.
PE	Program Counter Exception. Occurs whenever the host does a read program counter with exception. Cleared with read by host.
ME	Memory Access Exception. Occurs whenever a memory operation results in a memory access exception. Cleared when read by the host.
ICODE	Interrupt Code. Can be read and written by a compute engine or the host.
IV	Interrupt Valid. Set and read by the compute engine to indicate and interrupt to the host. Read and cleared by the host.

Figure 30

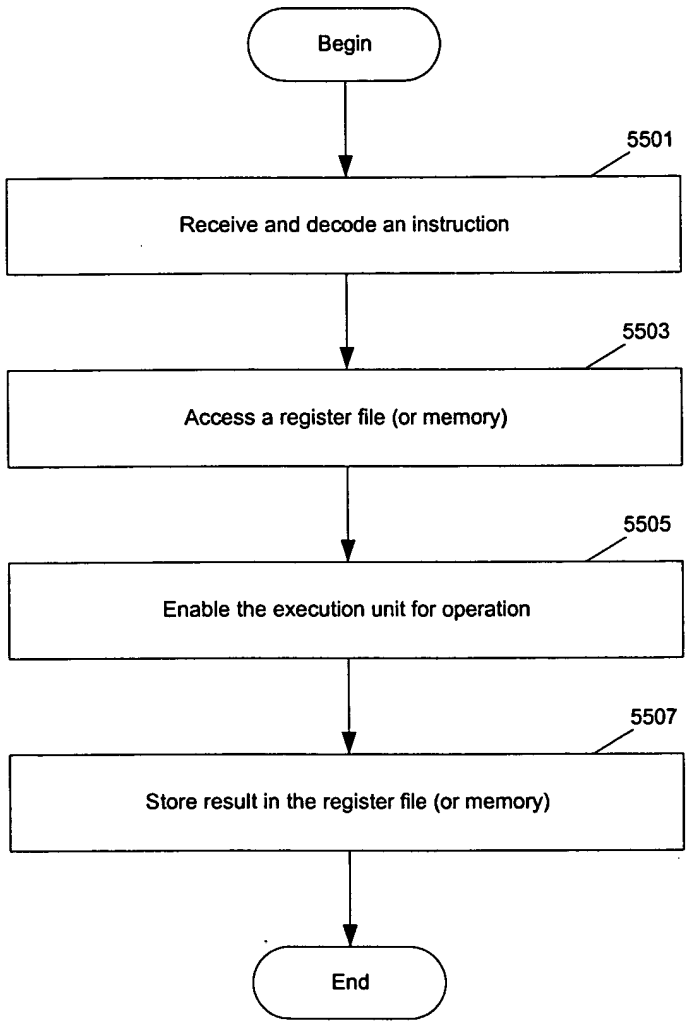

Begin
5501
Receive and decode an instruction
5503
Access a register file (or memory)
5505
Enable the execution unit for operation
5507
Store result in the register file (or memory)
End

Fig. 31

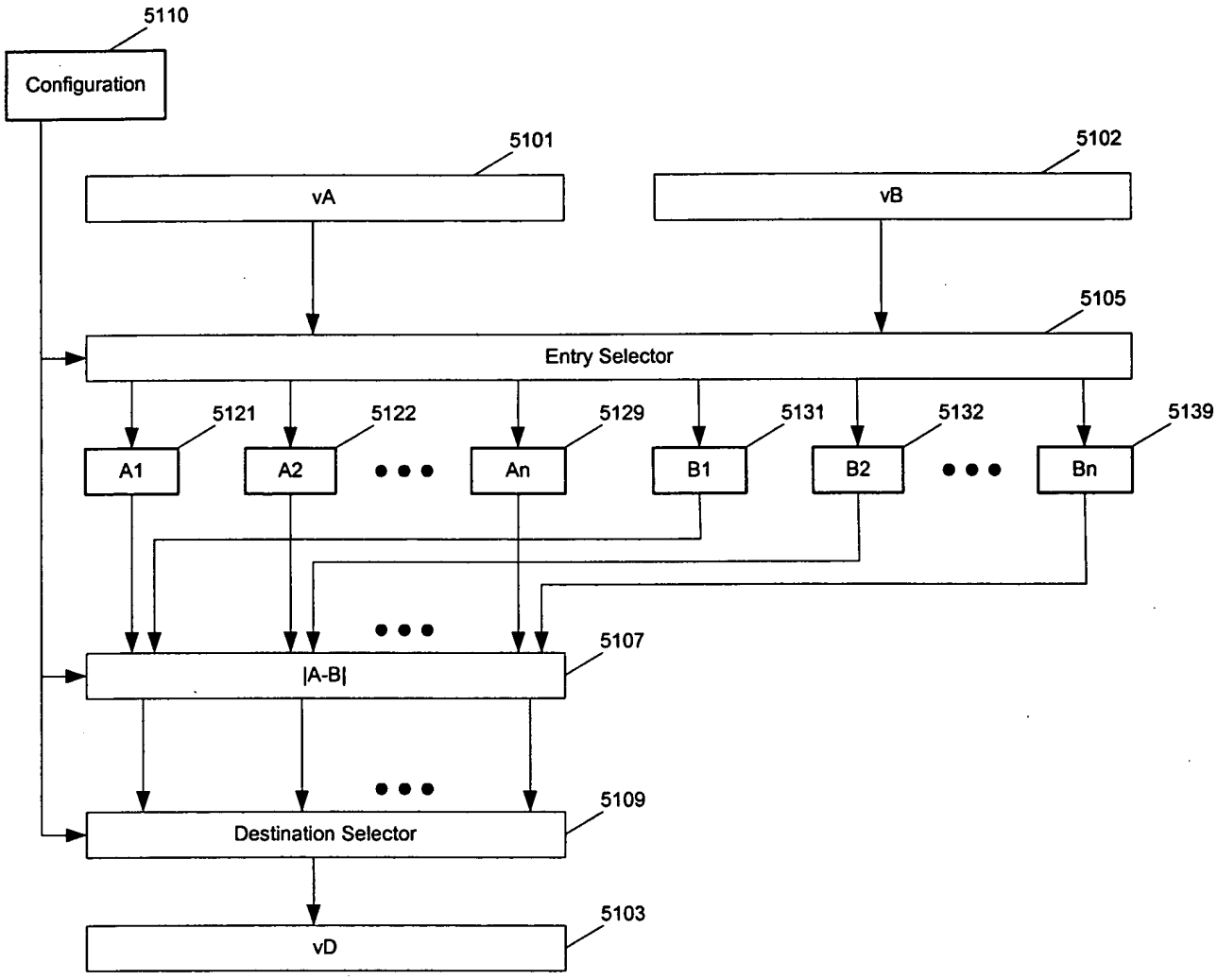

5110
Configuration
5101
vA
5102
vB
5105
Entry Selector
5121
A1
5122
A2
5129
An
5131
B1
5132
B2
5139
Bn
5107
|A-B|
5109
Destination Selector
5103
vD

Fig. 32

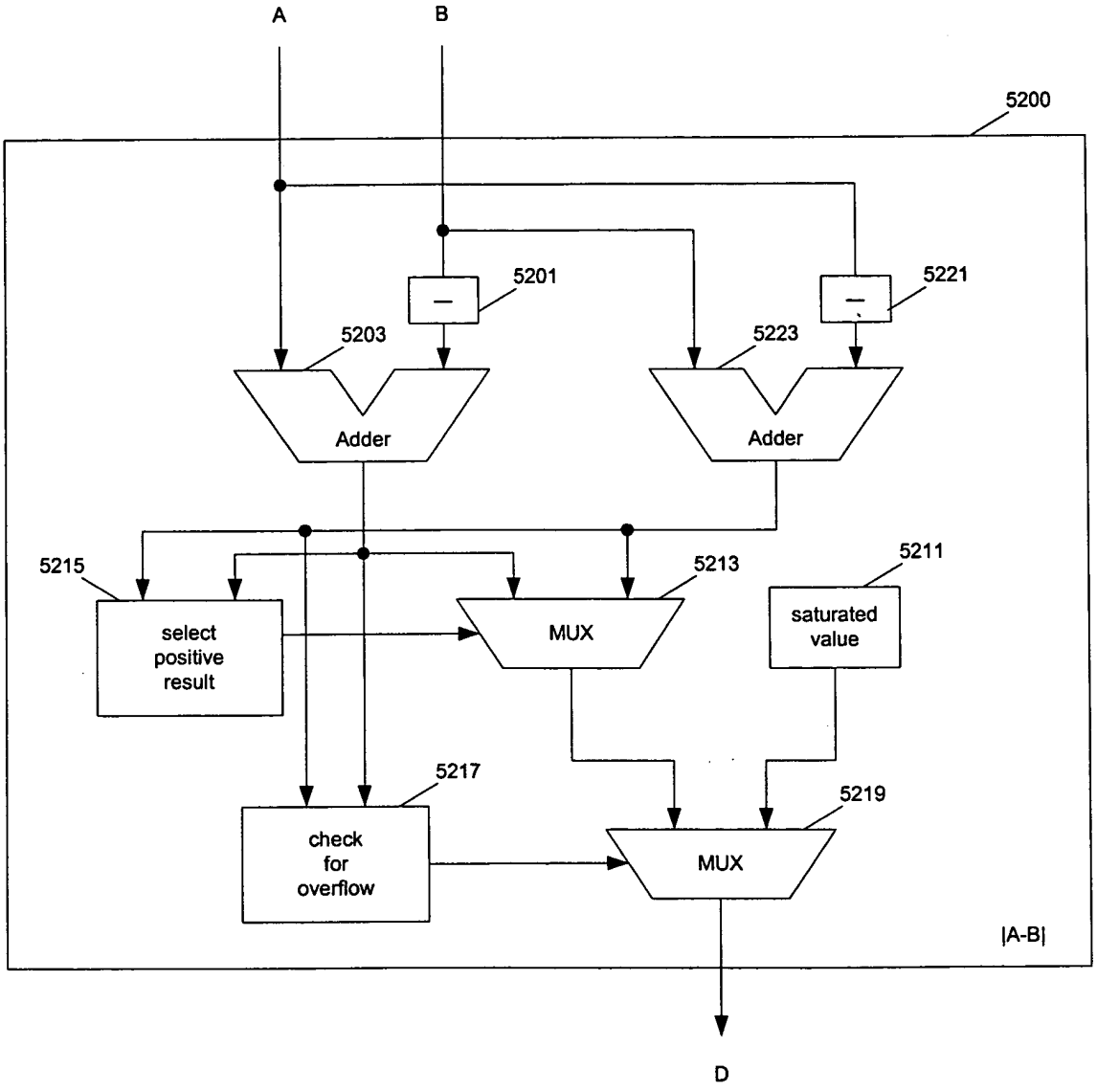
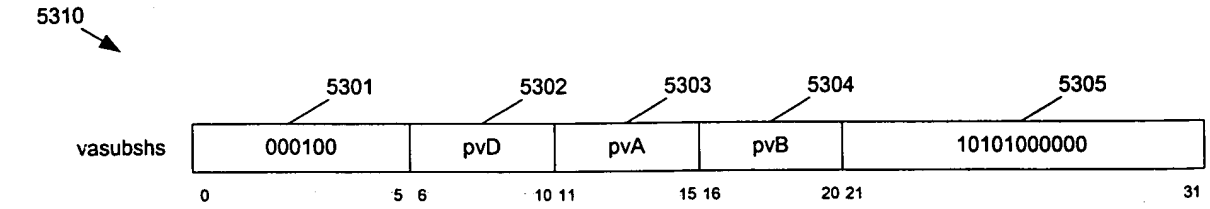

A
B
5200
5201
5221
5203
5223
Adder
Adder
5215
5213
5211
select positive result
MUX
saturated value
5217
5219
check for overflow
MUX
|A-B|
D

Fig. 33


5310
5301
5302
5303
5304
5305
vasubshs
000100
pvD
pvA
pvB
10101000000
0
5 6
10 11
15 16
20 21
31

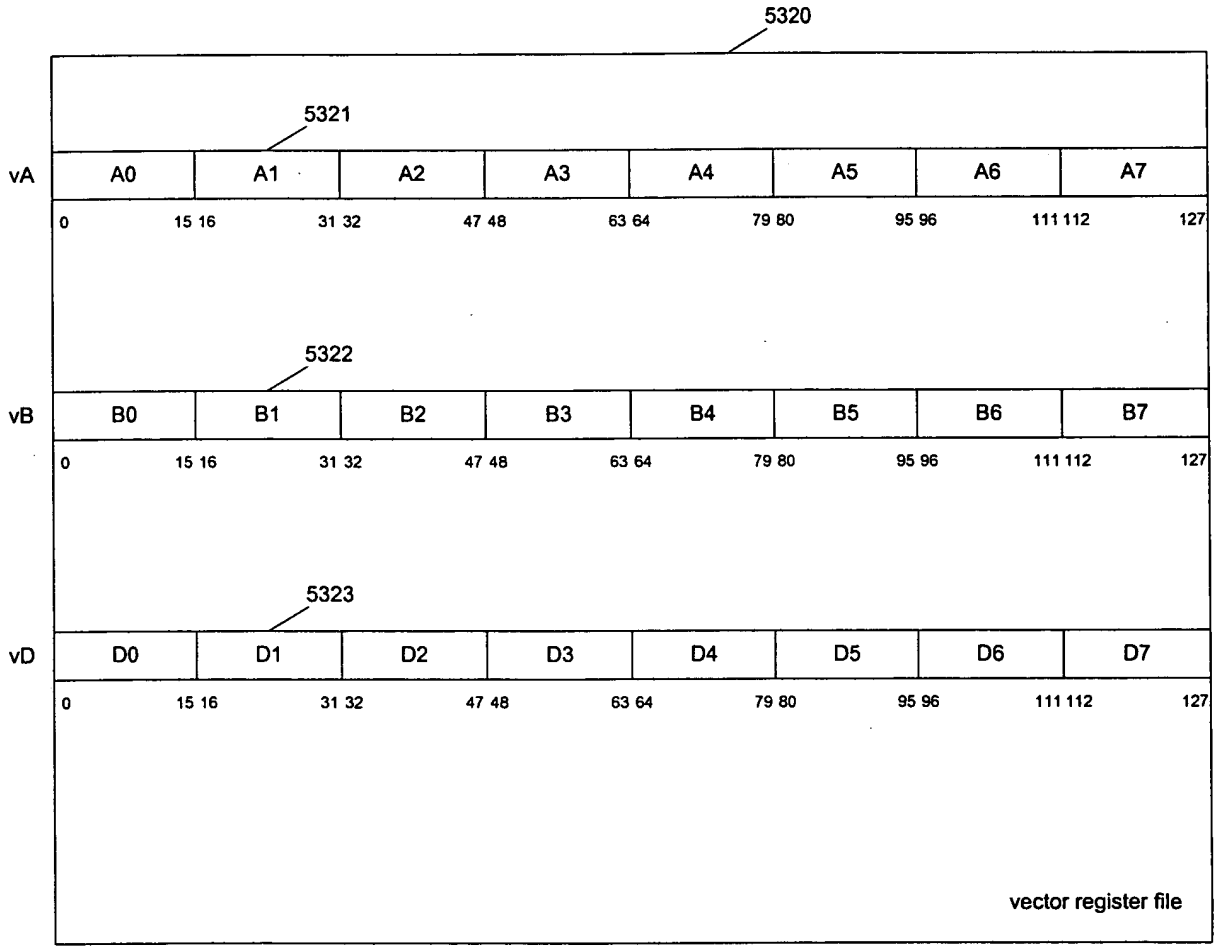

5320
5321
vA
A0
A1
A2
A3
A4
A5
A6
A7
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
5322
vB
B0
B1
B2
B3
B4
B5
B6
B7
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
5323
vD
D0
D1
D2
D3
D4
D5
D6
D7
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
vector register file

Fig. 34

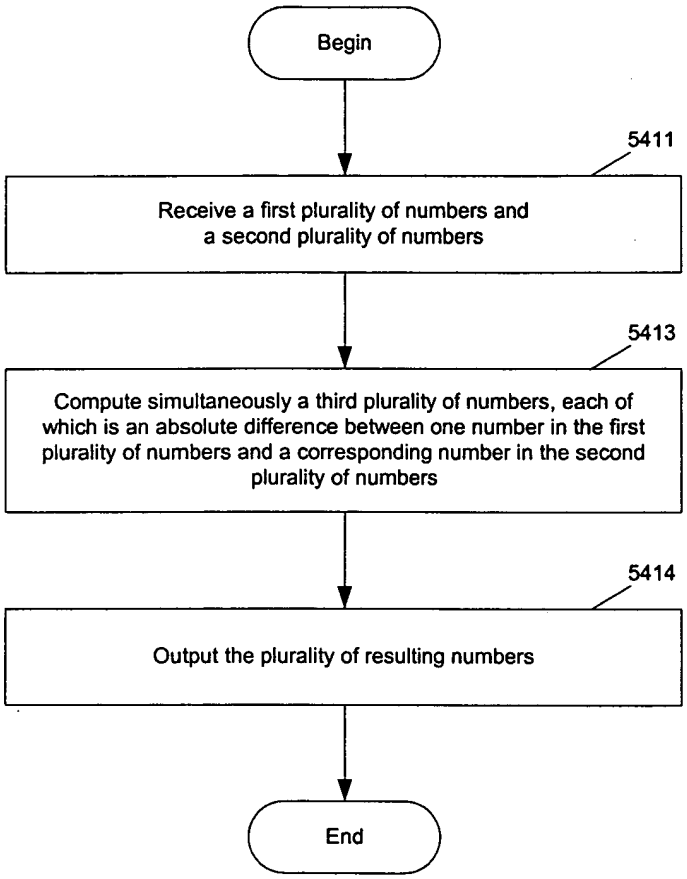

Begin
5411
Receive a first plurality of numbers and a second plurality of numbers
5413
Compute simultaneously a third plurality of numbers, each of which is an absolute difference between one number in the first plurality of numbers and a corresponding number in the second plurality of numbers
5414
Output the plurality of resulting numbers
End

Fig. 35

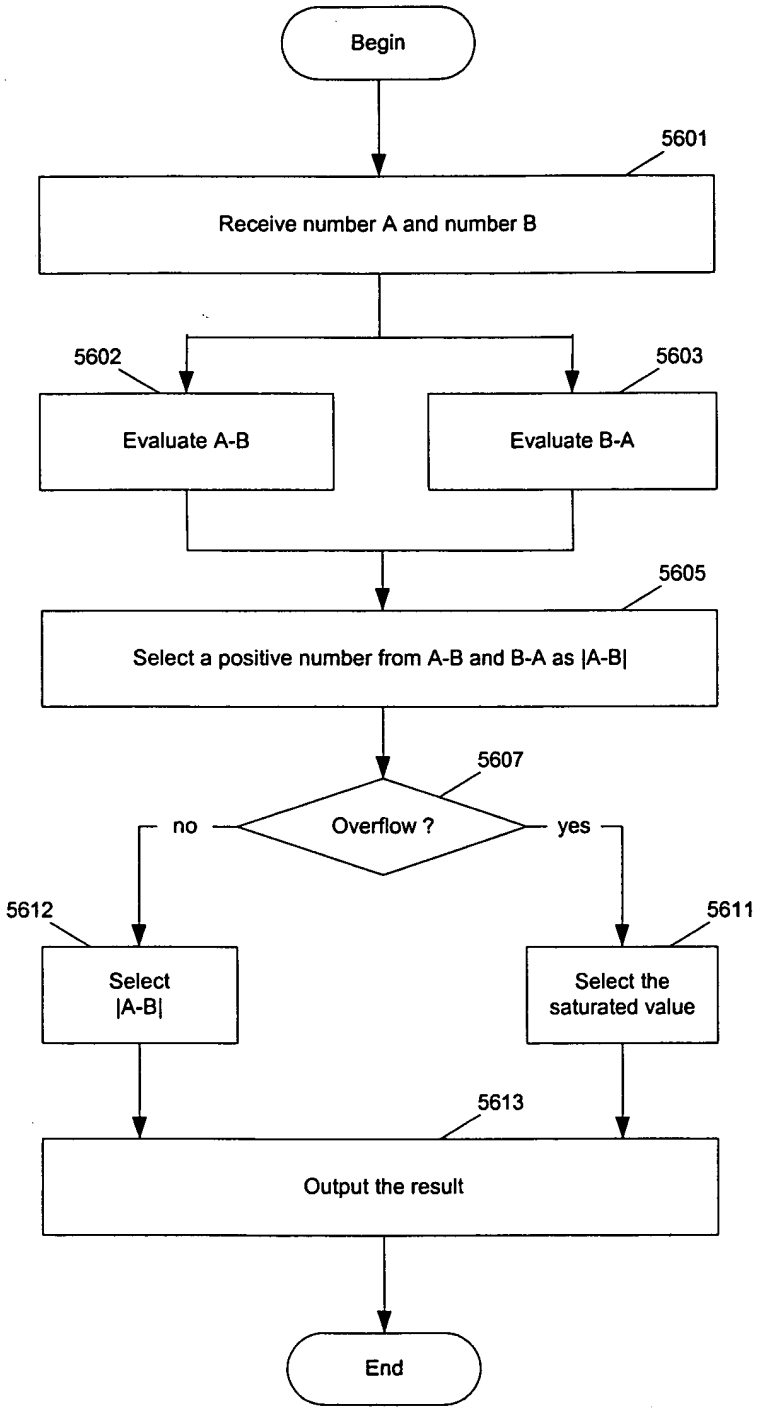

Begin
5601
Receive number A and number B
5602
5603
Evaluate A-B
Evaluate B-A
5605
Select a positive number from A-B and B-A as |A-B|
5607
Overflow ?
no
yes
5612
5611
Select
|A-B|
Select the
saturated value
5613
Output the result
End

Fig. 36

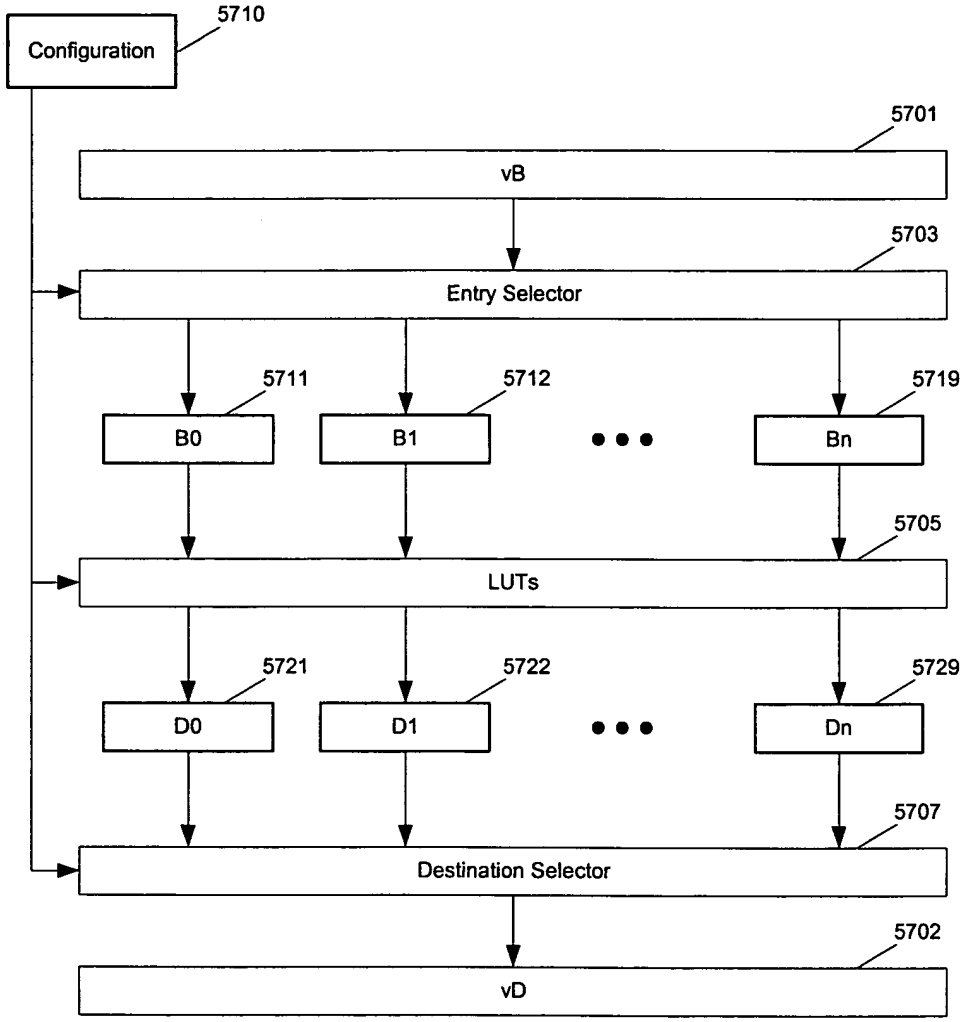

Configuration
5710
5701
vB
5703
Entry Selector
5711
5712
5719
B0
B1
Bn
5705
LUTs
5721
5722
5729
D0
D1
Dn
5707
Destination Selector
5702
vD

Fig. 37

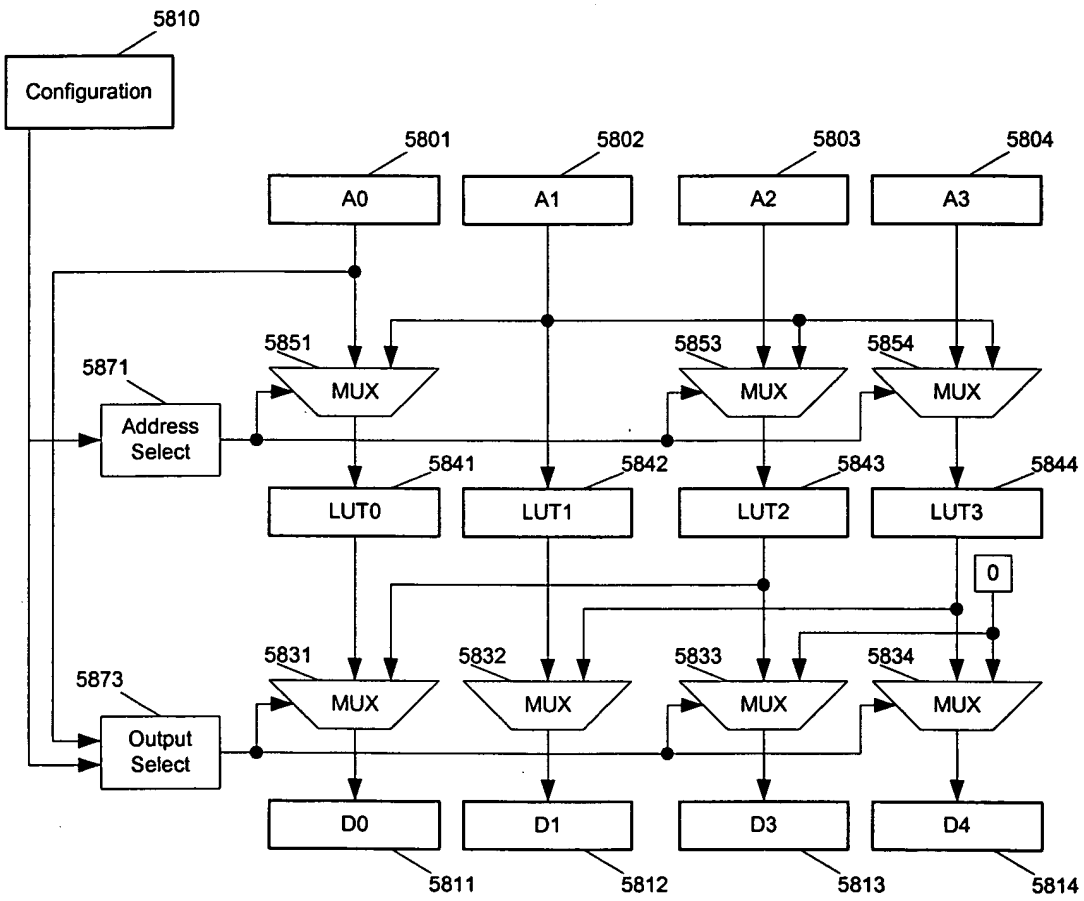

5810
Configuration
5801
5802
5803
5804
A0
A1
A2
A3
5851
5853
5854
5871
MUX
MUX
MUX
Address Select
5841
5842
5843
5844
LUT0
LUT1
LUT2
LUT3
0
5831
5832
5833
5834
5873
MUX
MUX
MUX
MUX
Output Select
D0
D1
D3
D4
5811
5812
5813
5814

Fig. 38

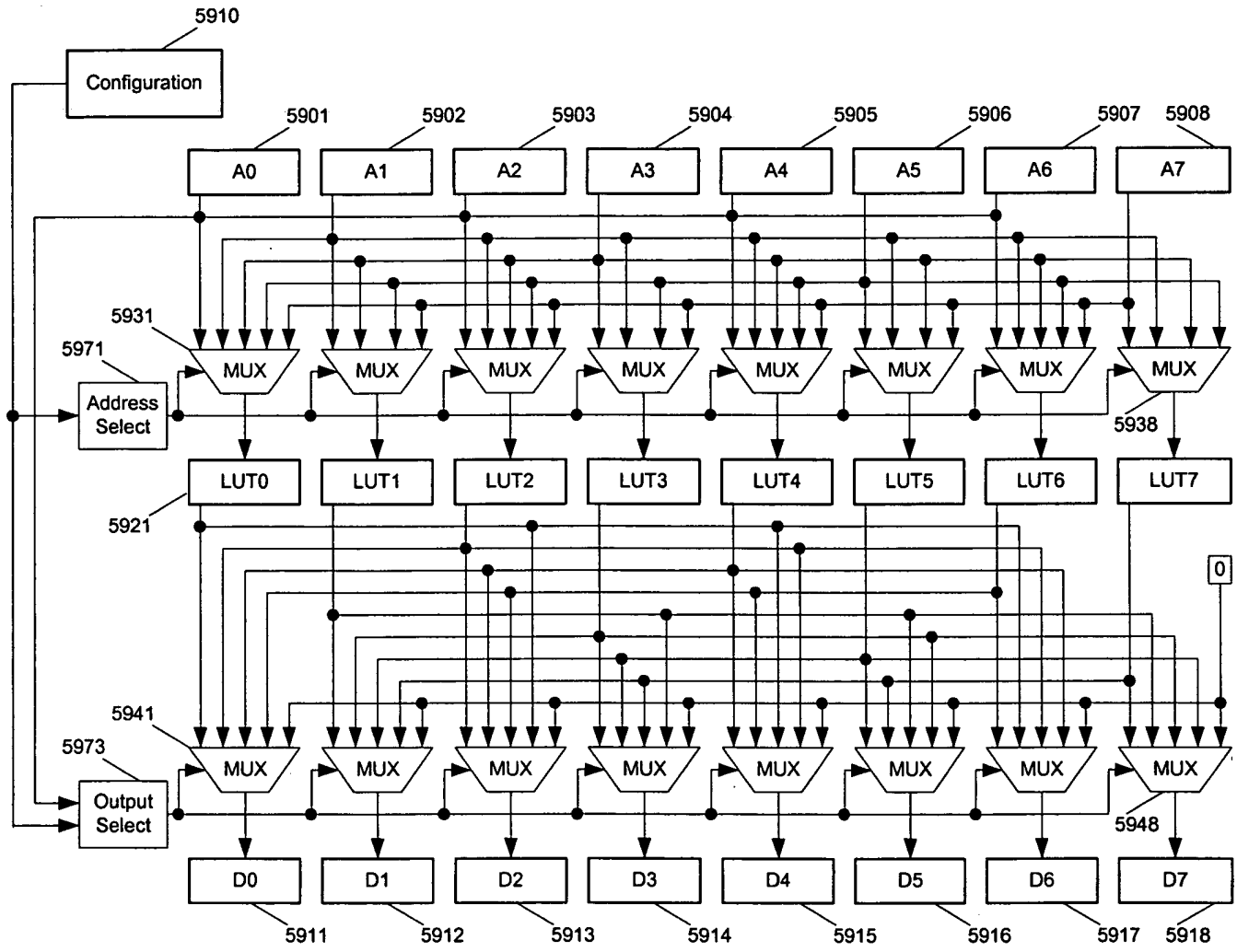

5910
Configuration
5901
5902
5903
5904
5905
5906
5907
5908
A0
A1
A2
A3
A4
A5
A6
A7
5931
5971
MUX
Address Select
5938
LUT0
LUT1
LUT2
LUT3
LUT4
LUT5
LUT6
LUT7
5921
0
5941
5973
Output Select
5948
D0
D1
D2
D3
D4
D5
D6
D7
5911
5912
5913
5914
5915
5916
5917
5918

Fig. 39

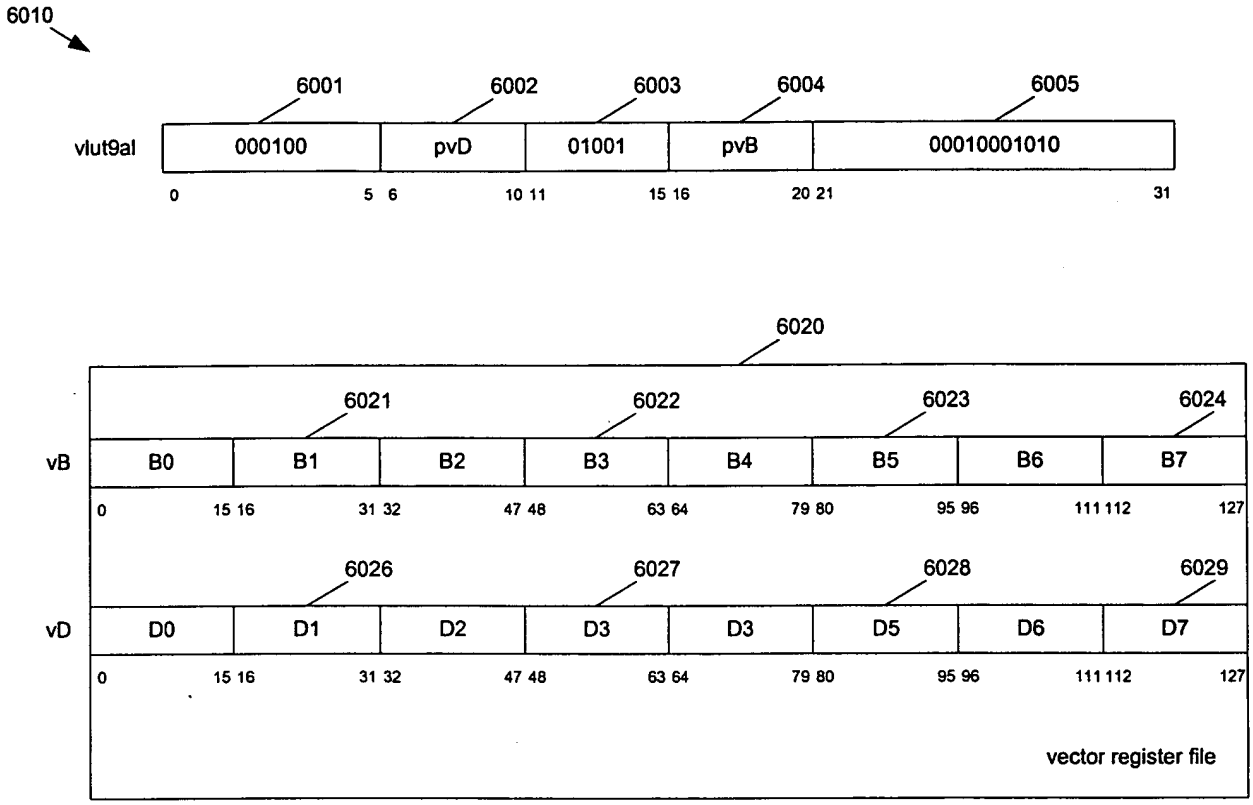

6010
6001
6002
6003
6004
6005
vlut9al
000100
pvD
01001
pvB
00010001010
0
5 6
10 11
15 16
20 21
31
6020
6021
6022
6023
6024
vB
B0
B1
B2
B3
B4
B5
B6
B7
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
6026
6027
6028
6029
vD
D0
D1
D2
D3
D3
D5
D6
D7
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
vector register file
6031
TAB0
6041
B1
D1
6032
TAB1
6042
B3
D3
6033
TAB2
6043
B5
D5
6034
TAB3
6044
B7
D7

Fig. 40

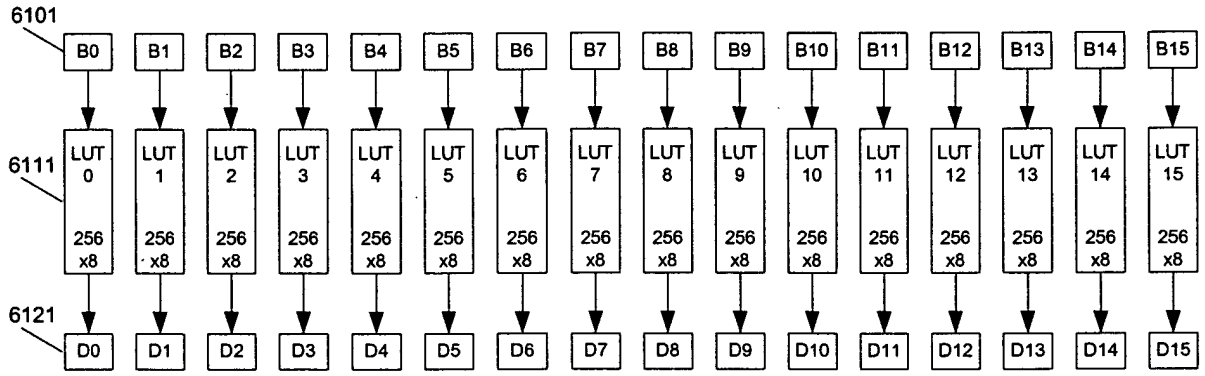

6101
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
6111
LUT 0
LUT 1
LUT 2
LUT 3
LUT 4
LUT 5
LUT 6
LUT 7
LUT 8
LUT 9
LUT 10
LUT 11
LUT 12
LUT 13
LUT 14
LUT 15
256 x8
6121
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15

Fig. 41

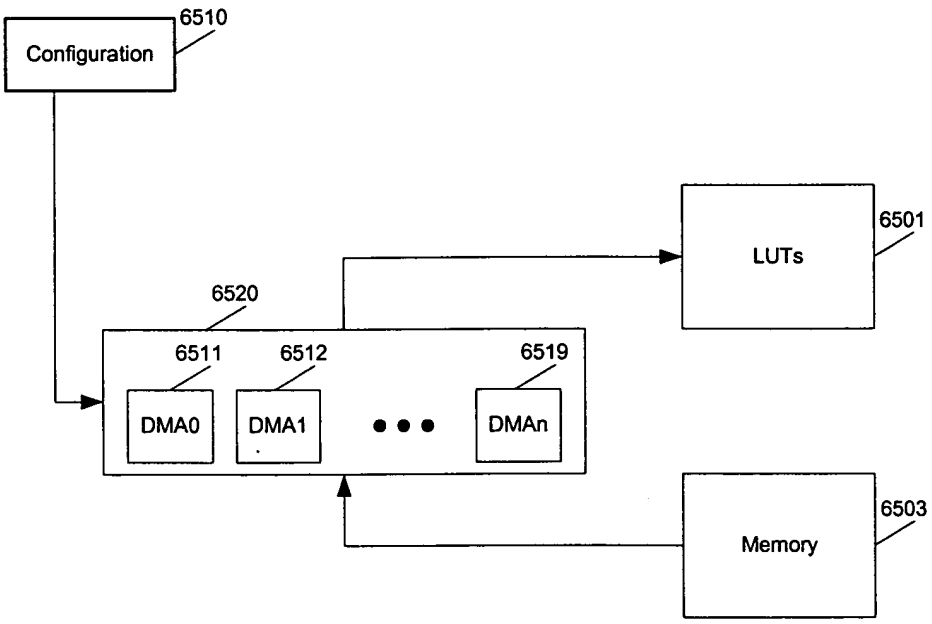

6510
Configuration
6501
LUTs
6520
6511
6512
6519
DMA0
DMA1
DMAn
6503
Memory

Fig. 45

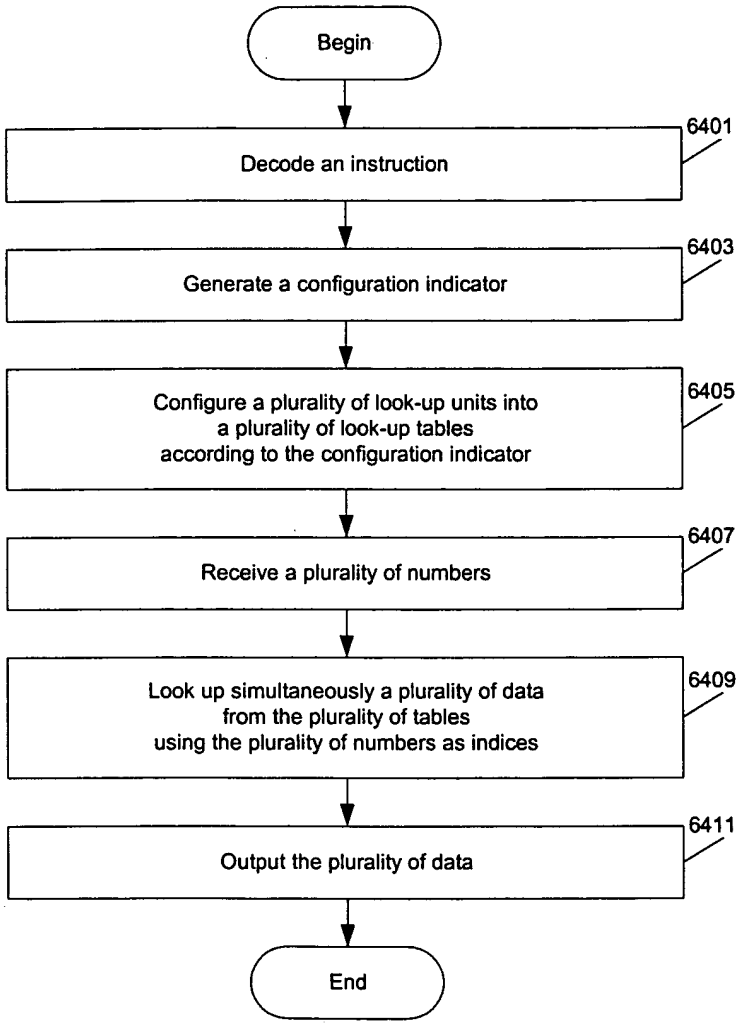

Begin
Decode an instruction
6401
Generate a configuration indicator
6403
Configure a plurality of look-up units into a plurality of look-up tables according to the configuration indicator
6405
Receive a plurality of numbers
6407
Look up simultaneously a plurality of data from the plurality of tables using the plurality of numbers as indices
6409
Output the plurality of data
6411
End

Fig. 44

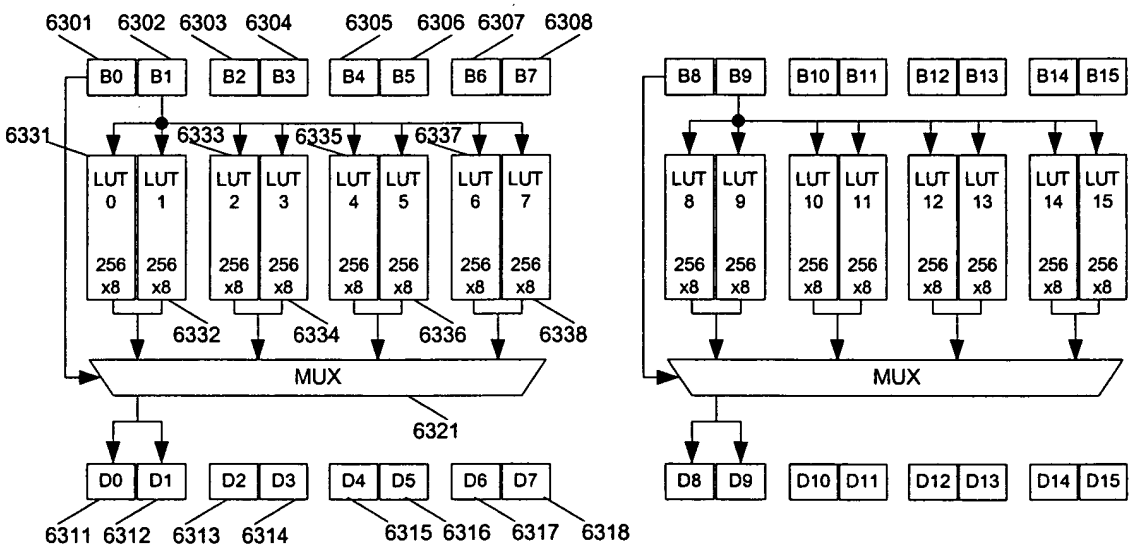

6301
6302
6303
6304
6305
6306
6307
6308
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
6331
6333
6335
6337
LUT 0
LUT 1
LUT 2
LUT 3
LUT 4
LUT 5
LUT 6
LUT 7
LUT 8
LUT 9
LUT 10
LUT 11
LUT 12
LUT 13
LUT 14
LUT 15
256 x8
6332
6334
6336
6338
MUX
6321
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
6311
6312
6313
6314
6315
6316
6317
6318

Fig. 43

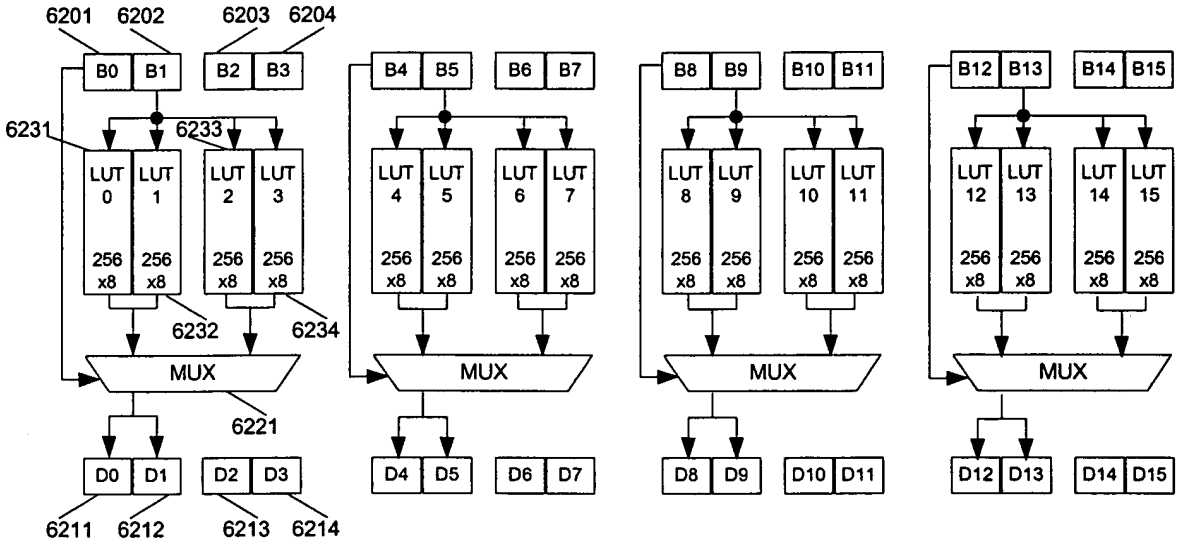
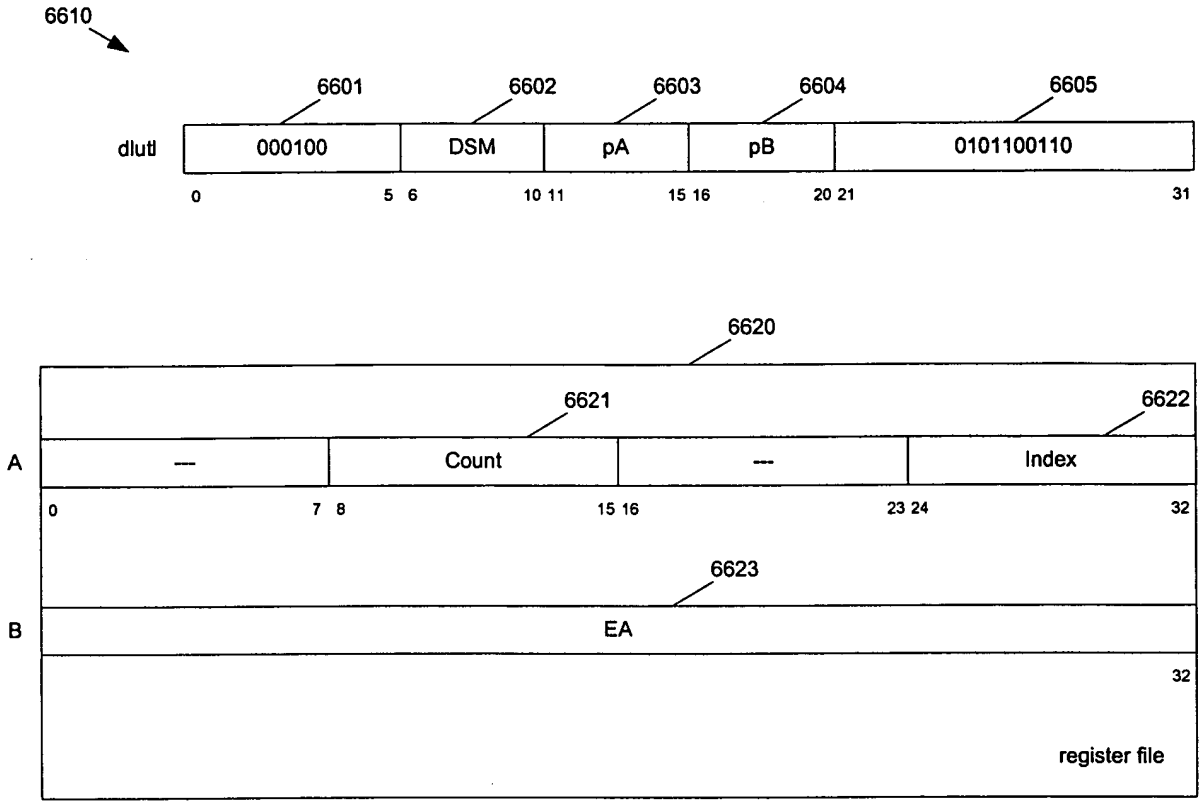

6201
6202
6203
6204
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
6231
6233
LUT 0
LUT 1
LUT 2
LUT 3
LUT 4
LUT 5
LUT 6
LUT 7
LUT 8
LUT 9
LUT 10
LUT 11
LUT 12
LUT 13
LUT 14
LUT 15
256 x8
6232
6234
MUX
6221
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
6211
6212
6213
6214

Fig. 42


6610
6601
6602
6603
6604
6605
dlutl
000100
DSM
pA
pB
0101100110
0
5 6
10 11
15 16
20 21
31
6620
6621
6622
A
—
Count
—
Index
0
7 8
15 16
23 24
32
6623
B
EA
32
register file

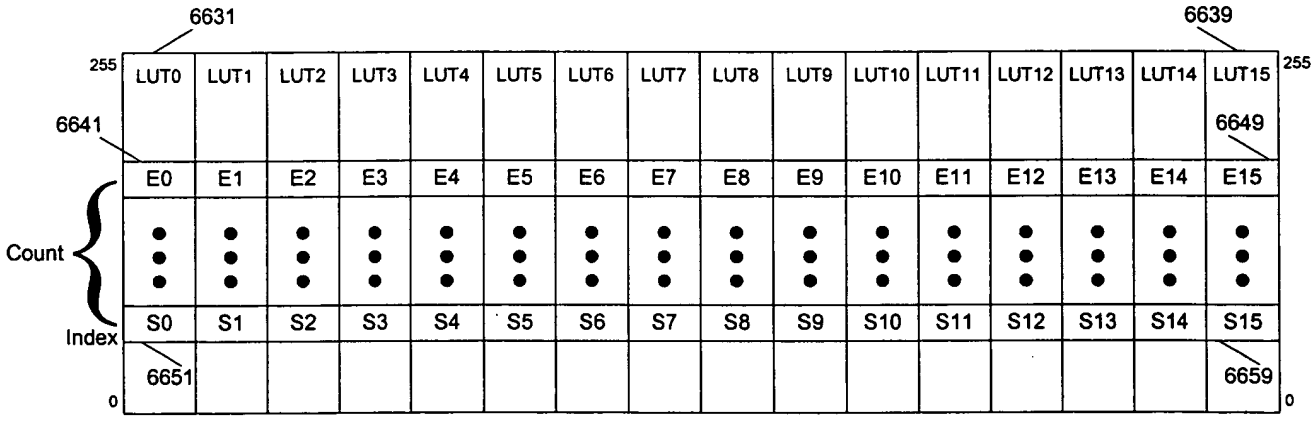

6631
6639
255
LUT0
LUT1
LUT2
LUT3
LUT4
LUT5
LUT6
LUT7
LUT8
LUT9
LUT10
LUT11
LUT12
LUT13
LUT14
LUT15
255
6641
6649
E0
E1
E2
E3
E4
E5
E6
E7
E8
E9
E10
E11
E12
E13
E14
E15
Count
Index
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
6651
6659
0
0

Fig. 46

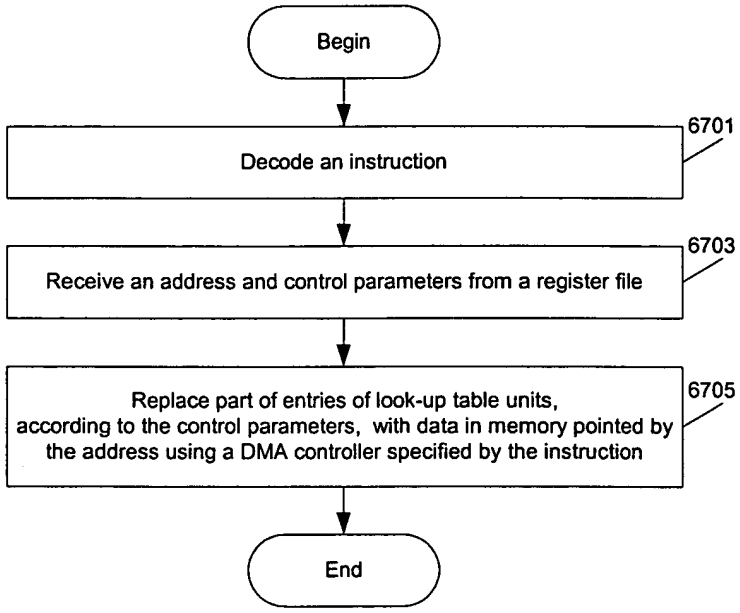

Begin
Decode an instruction
6701
Receive an address and control parameters from a register file
6703
Replace part of entries of look-up table units, according to the control parameters, with data in memory pointed by the address using a DMA controller specified by the instruction
6705
End

Fig. 47

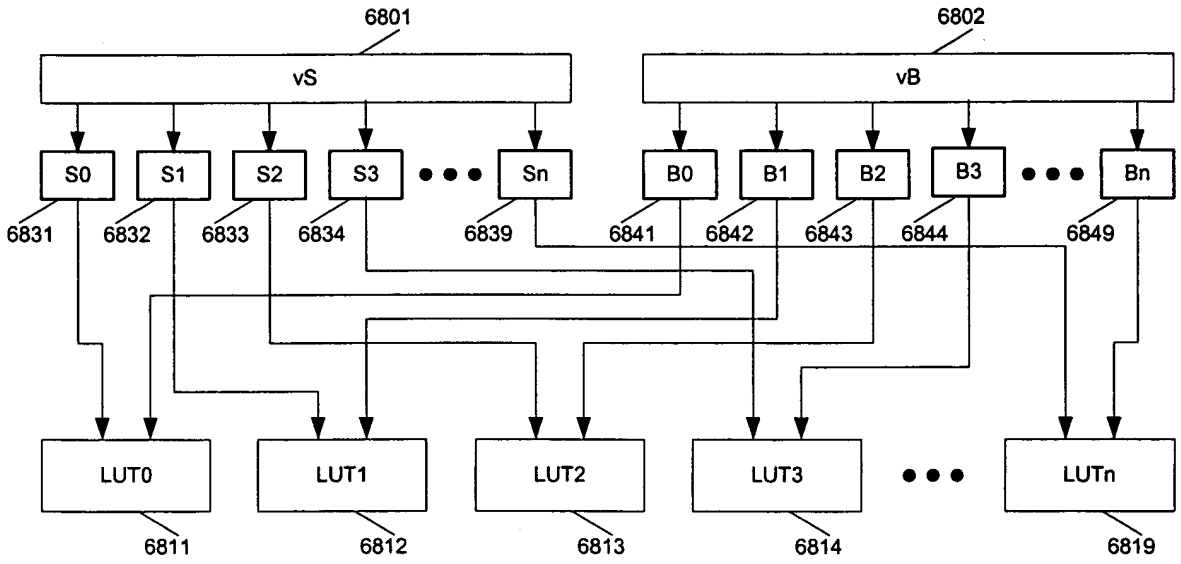
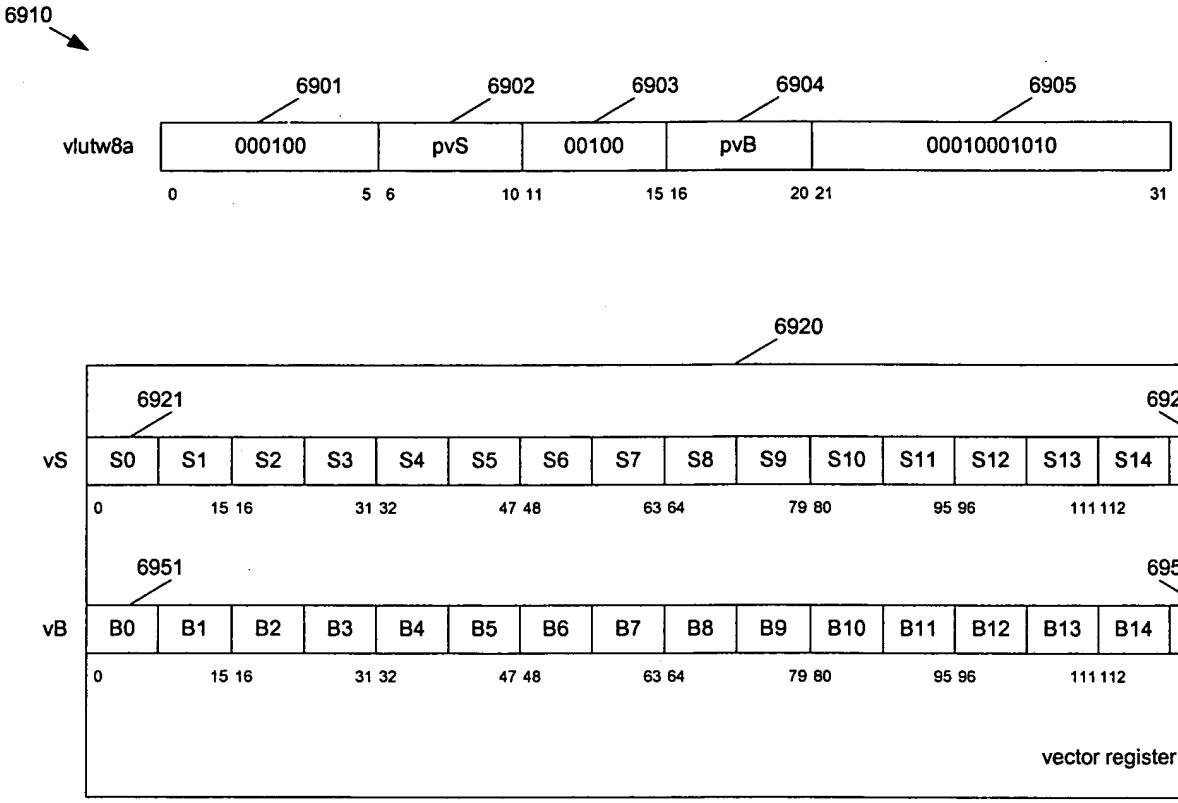

6801
6802
vS
vB
S0
S1
S2
S3
Sn
B0
B1
B2
B3
Bn
6831
6832
6833
6834
6839
6841
6842
6843
6844
6849
LUT0
LUT1
LUT2
LUT3
LUTn
6811
6812
6813
6814
6819

Fig. 48


6910
6901
6902
6903
6904
6905
vlutw8a
000100
pvS
00100
pvB
00010001010
0
5 6
10 11
15 16
20 21
31
6920
6921
6929
vS
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11
S12
S13
S14
S15
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
6951
6959
vB
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
0
15 16
31 32
47 48
63 64
79 80
95 96
111 112
127
vector register file

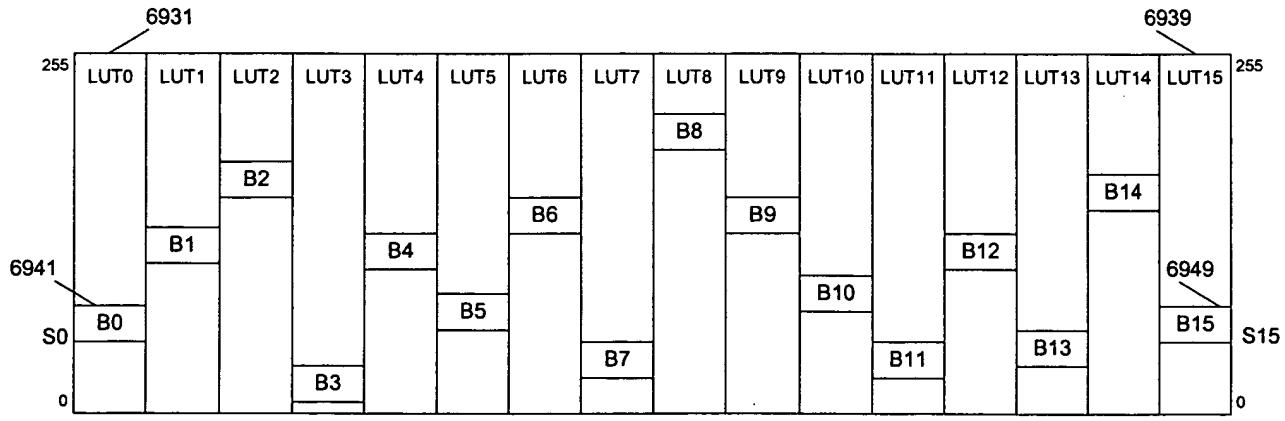

6931
6939
255
LUT0
LUT1
LUT2
LUT3
LUT4
LUT5
LUT6
LUT7
LUT8
LUT9
LUT10
LUT11
LUT12
LUT13
LUT14
LUT15
255
B8
B2
B14
B6
B9
B1
B12
B4
6941
B10
6949
B0
B5
B15
S0
B7
B11
B13
S15
B3
0
0

Fig. 49

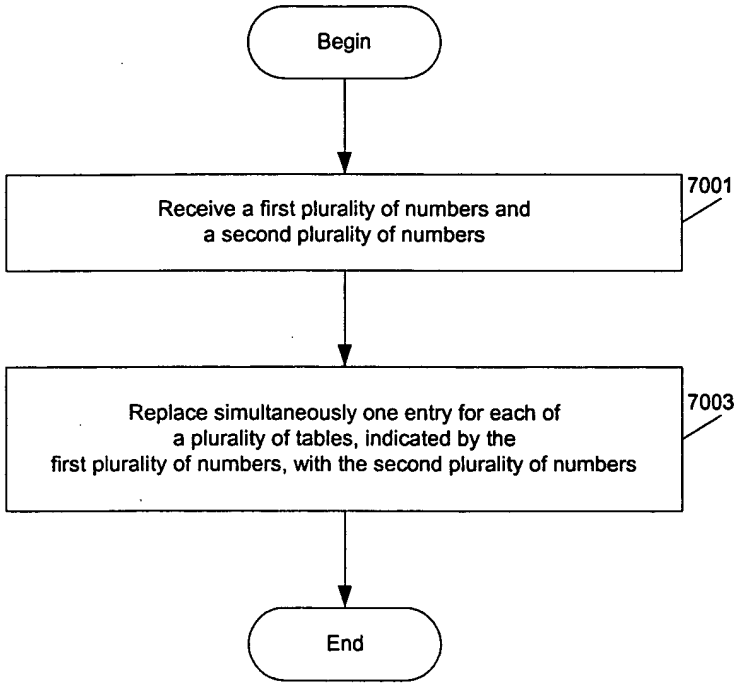

Begin
Receive a first plurality of numbers and a second plurality of numbers
7001
Replace simultaneously one entry for each of a plurality of tables, indicated by the first plurality of numbers, with the second plurality of numbers
7003
End

Fig. 50

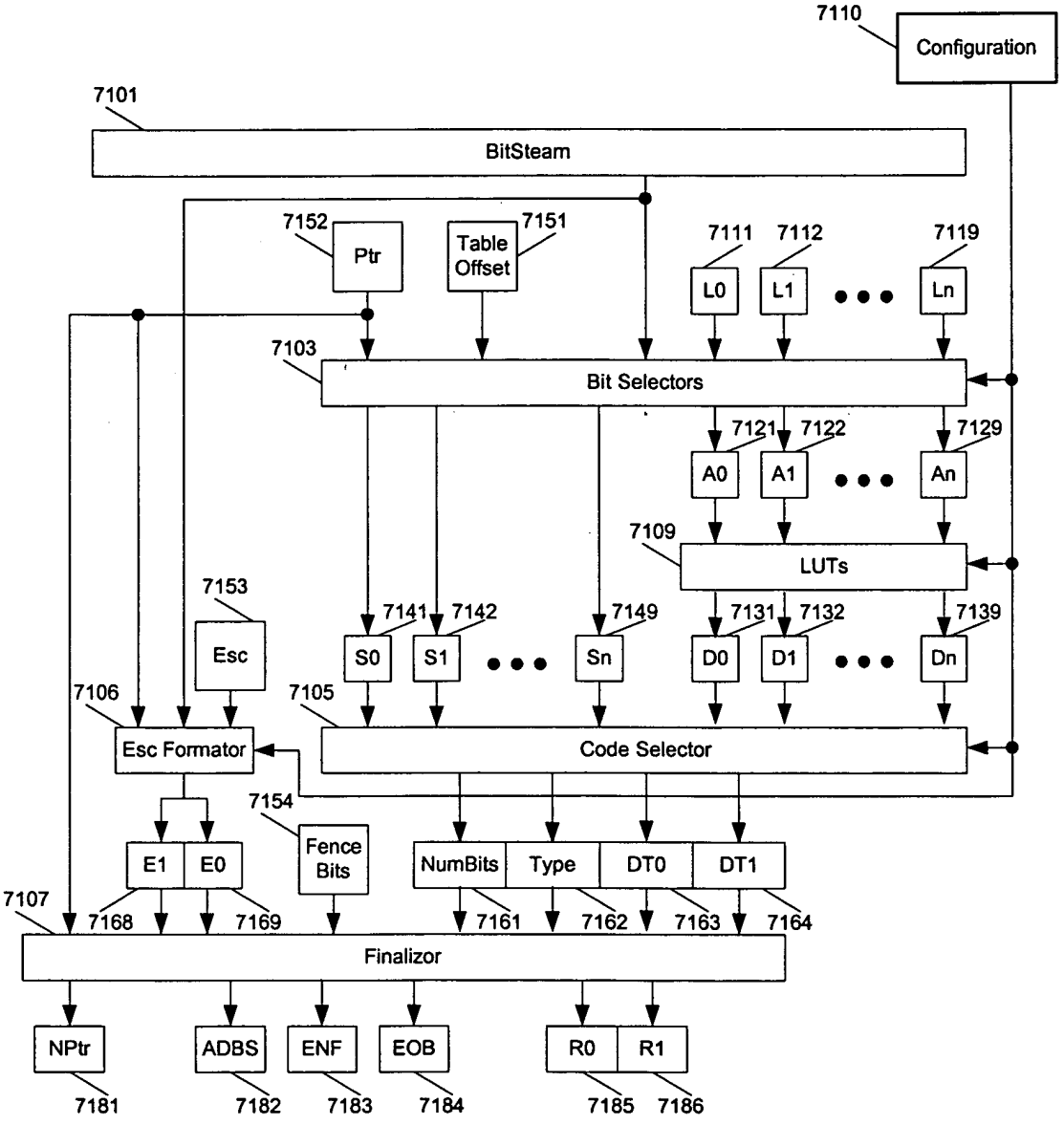

7110
Configuration
7101
BitSteam
7152
Ptr
7151
Table Offset
7111
7112
7119
L0
L1
Ln
7103
Bit Selectors
7121
7122
7129
A0
A1
An
7109
LUTs
7153
Esc
7141
7142
7149
7131
7132
7139
S0
S1
Sn
D0
D1
Dn
7106
7105
Esc Formator
Code Selector
7154
Fence Bits
E1
E0
NumBits
Type
DT0
DT1
7107
7168
7169
7161
7162
7163
7164
Finalizor
NPtr
ADBS
ENF
EOB
R0
R1
7181
7182
7183
7184
7185
7186

Fig. 51

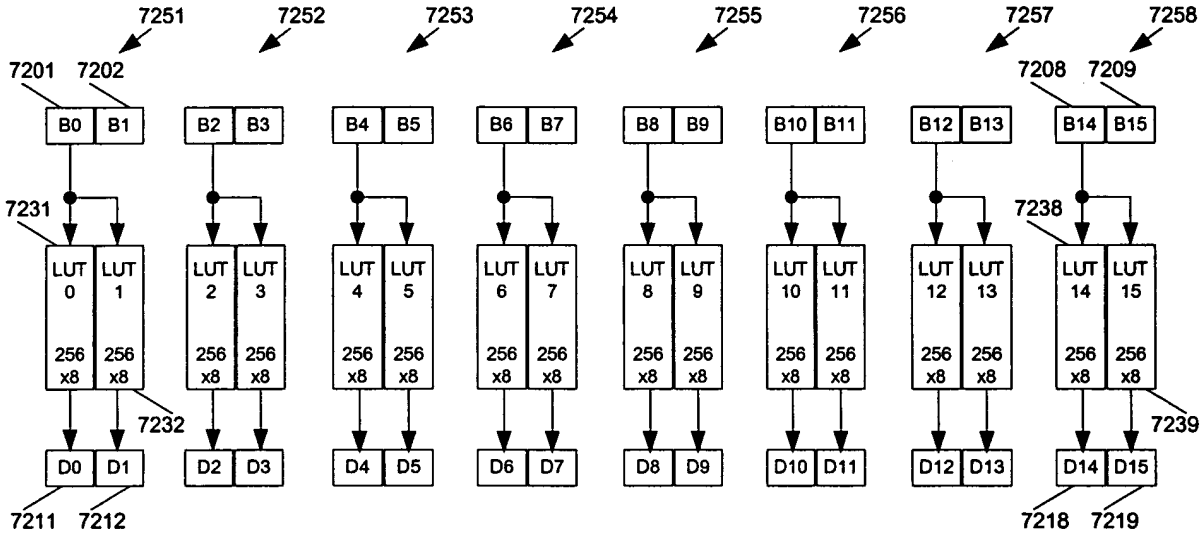

7251
7252
7253
7254
7255
7256
7257
7258
7201
7202
7208
7209
B0
B1
B2
B3
B4
B5
B6
B7
B8
B9
B10
B11
B12
B13
B14
B15
7231
7238
LUT 0
LUT 1
LUT 2
LUT 3
LUT 4
LUT 5
LUT 6
LUT 7
LUT 8
LUT 9
LUT 10
LUT 11
LUT 12
LUT 13
LUT 14
LUT 15
256 x8
7232
7239
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15
7211
7212
7218
7219

Fig. 52

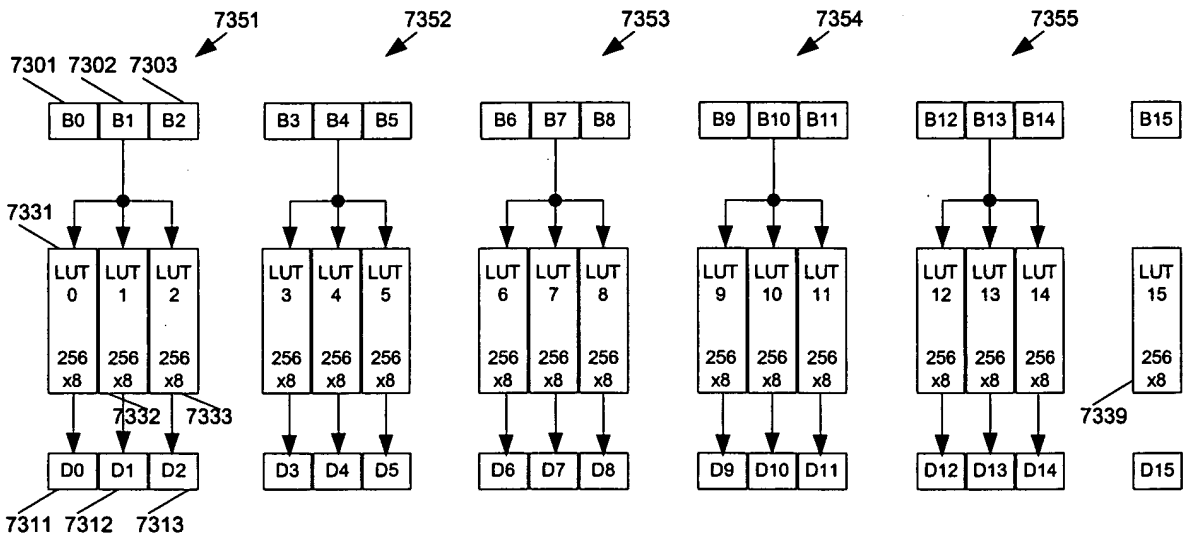

7351
7352
7353
7354
7355
7301 7302 7303
B0 B1 B2
B3 B4 B5
B6 B7 B8
B9 B10 B11
B12 B13 B14
B15
7331
LUT 0
LUT 1
LUT 2
LUT 3
LUT 4
LUT 5
LUT 6
LUT 7
LUT 8
LUT 9
LUT 10
LUT 11
LUT 12
LUT 13
LUT 14
LUT 15
256 x8
7332
7333
7339
D0 D1 D2
D3 D4 D5
D6 D7 D8
D9 D10 D11
D12 D13 D14
D15
7311 7312 7313

Fig. 53

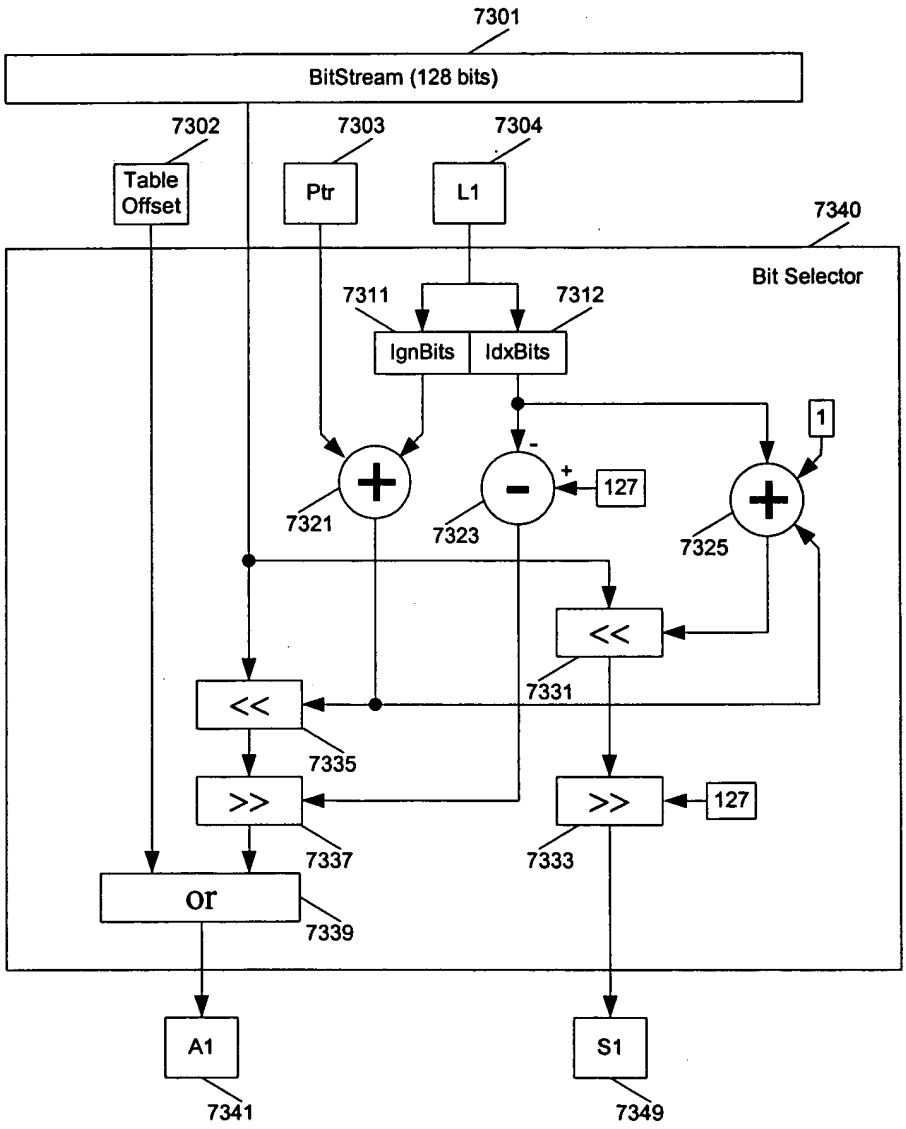

7301
BitStream (128 bits)
7302
Table Offset
7303
Ptr
7304
L1
7340
Bit Selector
7311
7312
IgnBits
IdxBits
1
7321
7323
127
7325
<<
7331
<<
7335
>>
7337
>>
127
7333
or
7339
A1
S1
7341
7349

Fig. 54

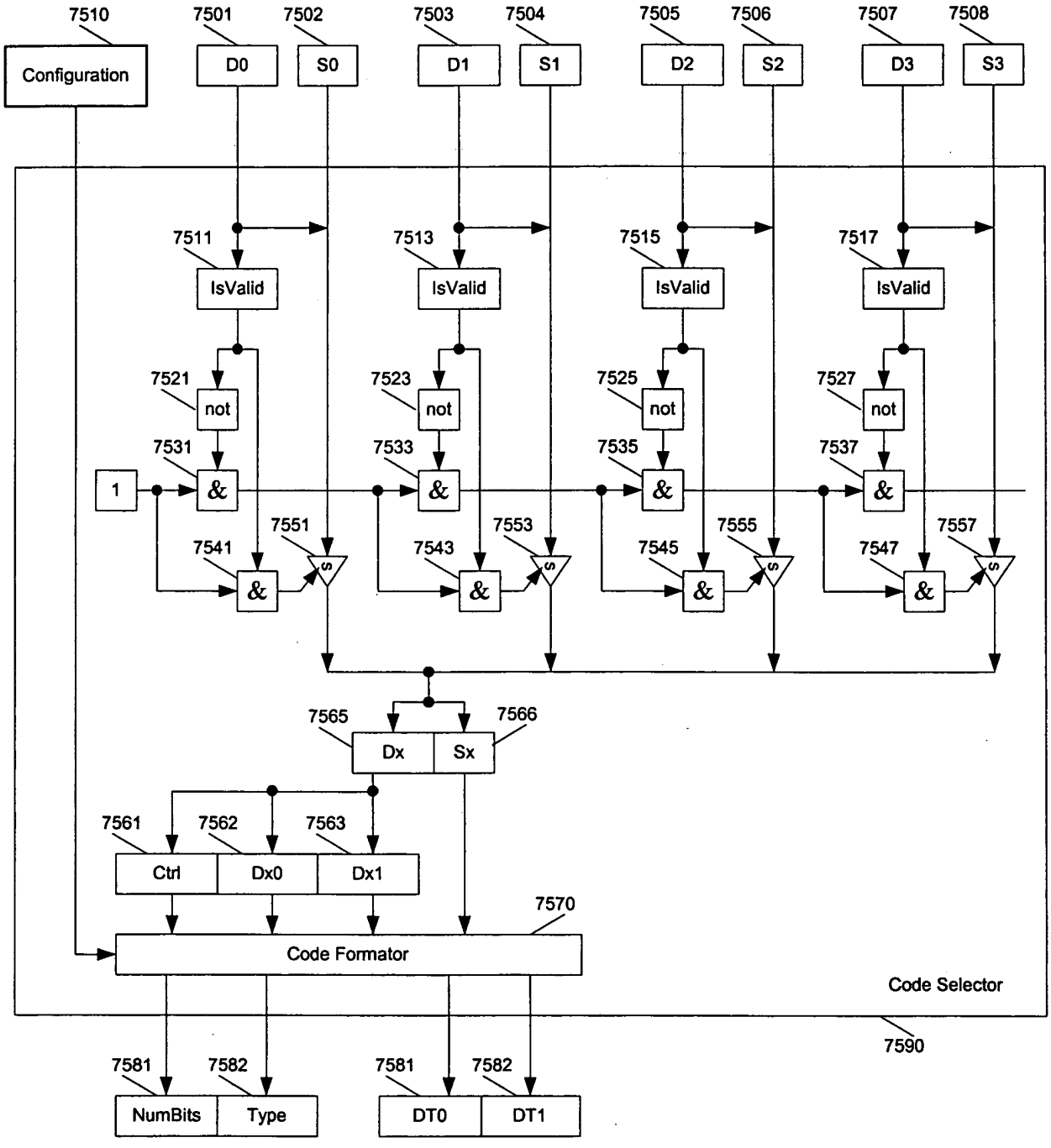

7510
Configuration
7501
D0
7502
S0
7503
D1
7504
S1
7505
D2
7506
S2
7507
D3
7508
S3
7511
IsValid
7513
IsValid
7515
IsValid
7517
IsValid
7521
not
7523
not
7525
not
7527
not
7531
7533
7535
7537
1
&
7541
7543
7545
7547
7551
7553
7555
7557
S
7565
Dx
7566
Sx
7561
Ctrl
7562
Dx0
7563
Dx1
7570
Code Formator
Code Selector
7590
7581
NumBits
7582
Type
7581
DT0
7582
DT1

Fig. 55

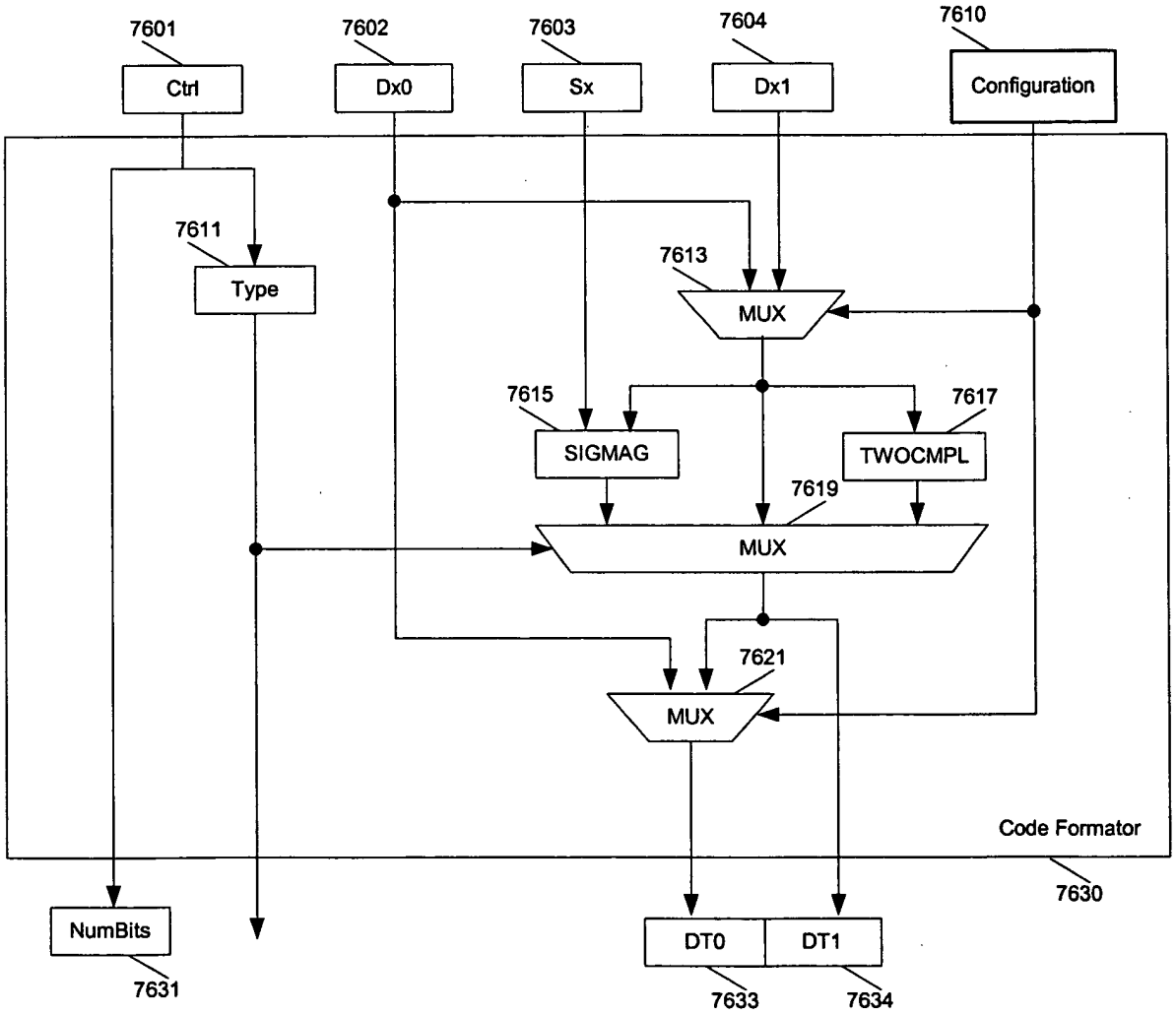

7601
Ctrl
7602
Dx0
7603
Sx
7604
Dx1
7610
Configuration
7611
Type
7613
MUX
7615
SIGMAG
7617
TWOCMPL
7619
MUX
7621
MUX
Code Formator
7630
NumBits
7631
DT0
DT1
7633
7634

Fig. 56

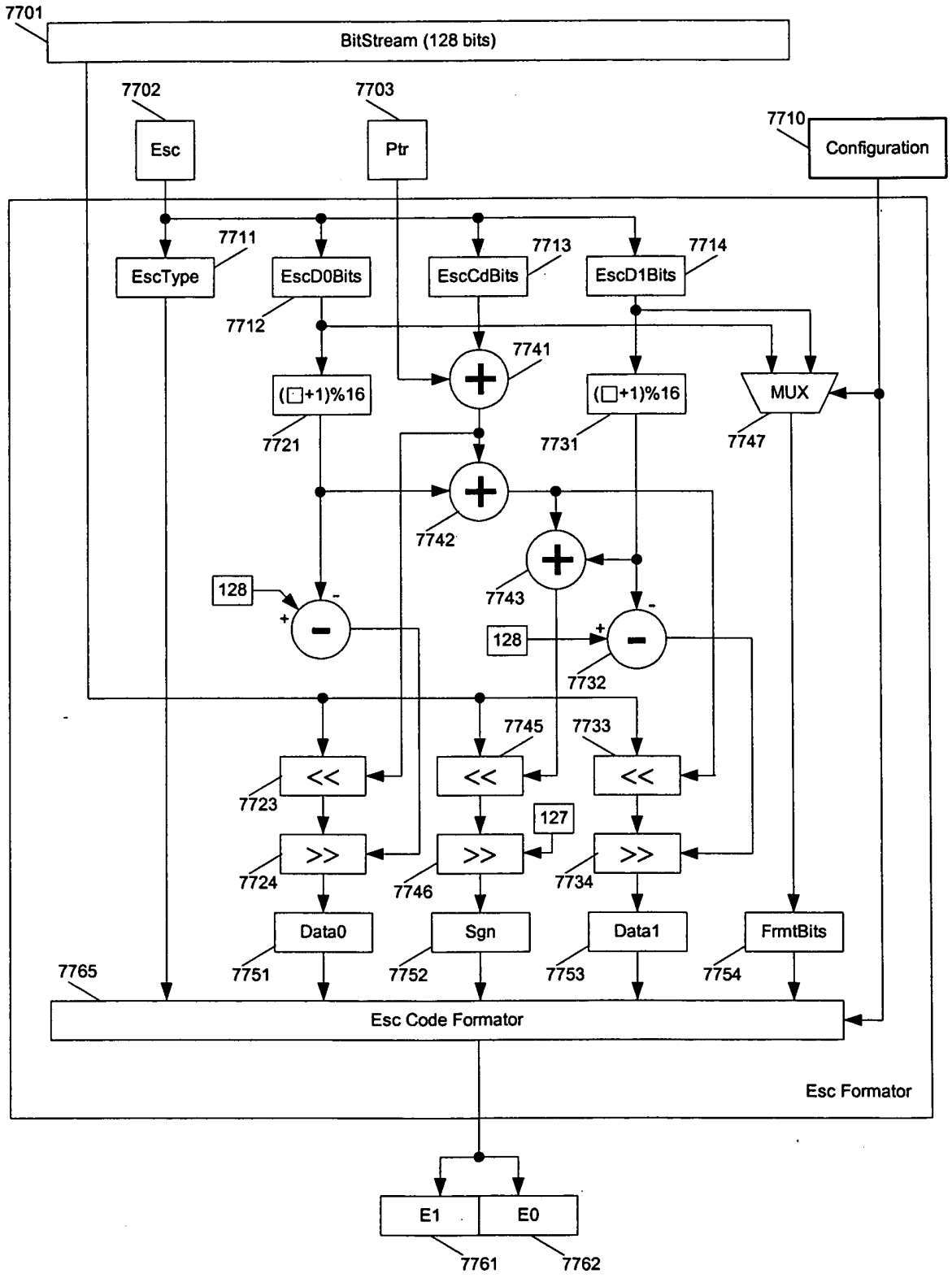

7701
BitStream (128 bits)
7702
Esc
7703
Ptr
7710
Configuration
7711
EscType
EscD0Bits
7712
EscCdBits
7713
EscD1Bits
7714
7741
(□+1)%16
7721
(□+1)%16
7731
MUX
7747
7742
128
7743
128
7732
7745
7733
<<
<<
<<
7723
127
>>
>>
>>
7724
7746
7734
Data0
Sgn
Data1
FrmtBits
7765
7751
7752
7753
7754
Esc Code Formator
Esc Formator
E1
E0
7761
7762

Fig. 57

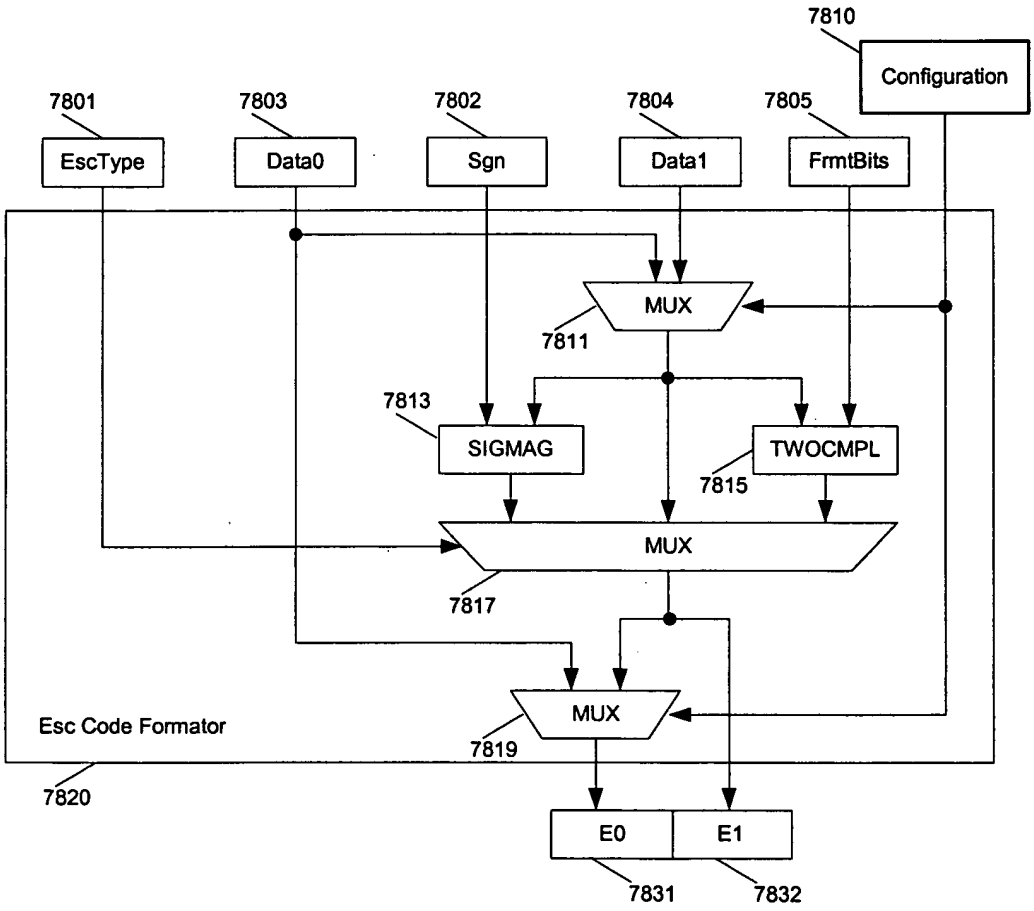

7810
Configuration
7801
EscType
7803
Data0
7802
Sgn
7804
Data1
7805
FrmtBits
MUX
7811
7813
SIGMAG
TWOCMPL
7815
MUX
7817
MUX
7819
Esc Code Formator
7820
E0
E1
7831
7832

Fig. 58

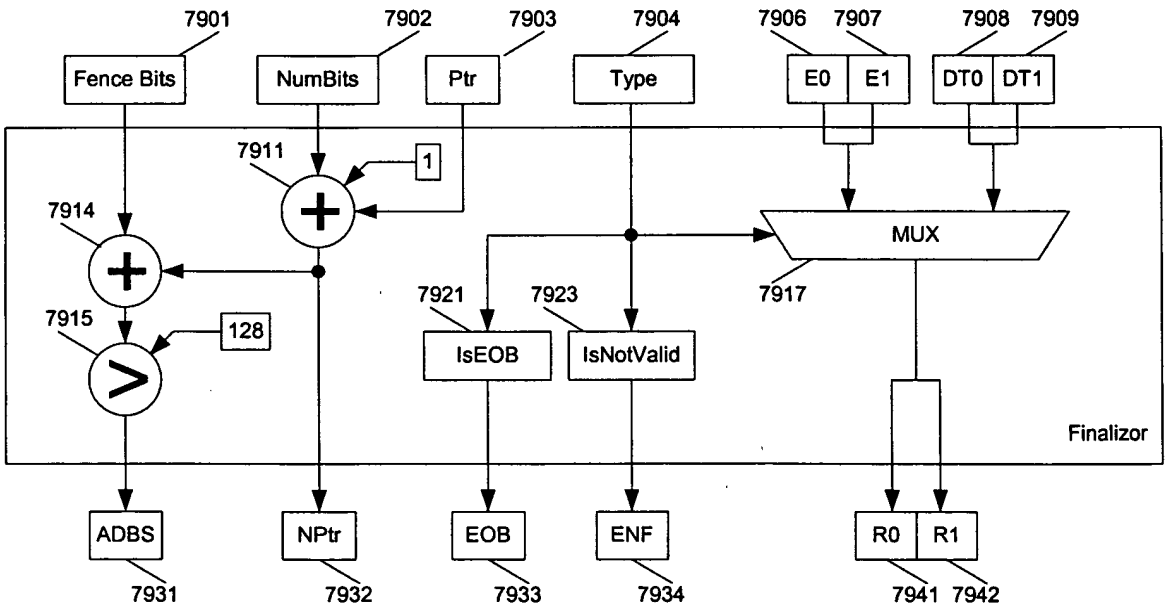

7901
7902
7903
7904
7906
7907
7908
7909
Fence Bits
NumBits
Ptr
Type
E0
E1
DT0
DT1
7911
1
7914
MUX
7915
128
7921
7923
7917
IsEOB
IsNotValid
Finalizor
ADBS
NPtr
EOB
ENF
R0
R1
7931
7932
7933
7934
7941
7942

Fig. 59

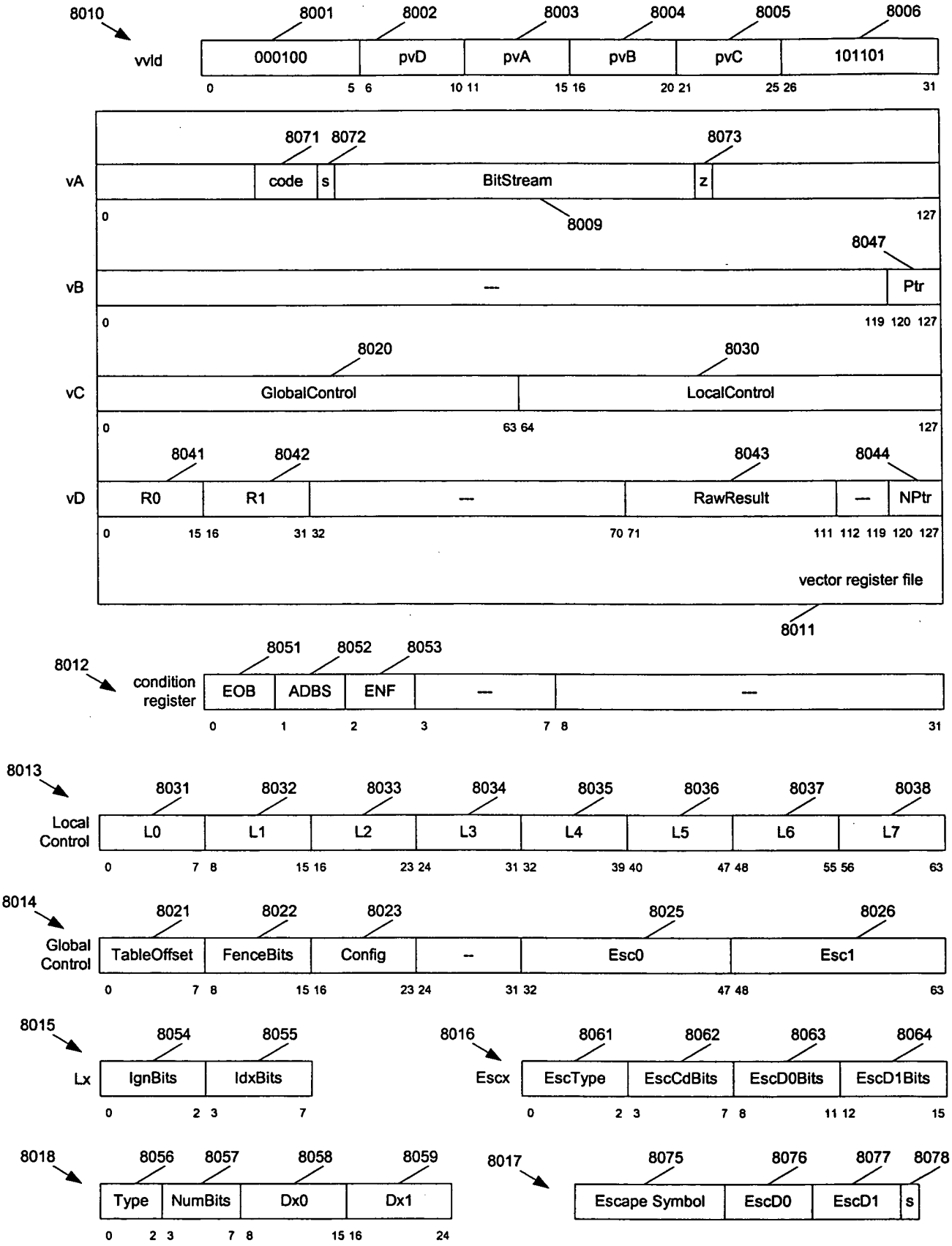

8010
vvld
8001
8002
8003
8004
8005
8006
000100
pvD
pvA
pvB
pvC
101101
0
5 6
10 11
15 16
20 21
25 26
31
vA
8071
8072
8073
code
s
BitStream
z
8009
0
127
vB
8047
—
Ptr
0
119 120 127
vC
8020
8030
GlobalControl
LocalControl
0
63 64
127
vD
8041
8042
8043
8044
R0
R1
—
RawResult
—
NPtr
0
15 16
31 32
70 71
111 112 119 120 127
vector register file
8011
8012
condition register
8051
8052
8053
EOB
ADBS
ENF
—
—
0
1
2
3
7 8
31
8013
Local Control
8031
8032
8033
8034
8035
8036
8037
8038
L0
L1
L2
L3
L4
L5
L6
L7
0
7 8
15 16
23 24
31 32
39 40
47 48
55 56
63
8014
Global Control
8021
8022
8023
8025
8026
TableOffset
FenceBits
Config
—
Esc0
Esc1
0
7 8
15 16
23 24
31 32
47 48
63
8015
Lx
8054
8055
IgnBits
IdxBits
0
2 3
7
8016
Escx
8061
8062
8063
8064
EscType
EscCdBits
EscD0Bits
EscD1Bits
0
2 3
7 8
11 12
15
8018
8056
8057
8058
8059
Type
NumBits
Dx0
Dx1
0
2 3
7 8
15 16
24
8017
8075
8076
8077
8078
Escape Symbol
EscD0
EscD1
s

Fig. 60

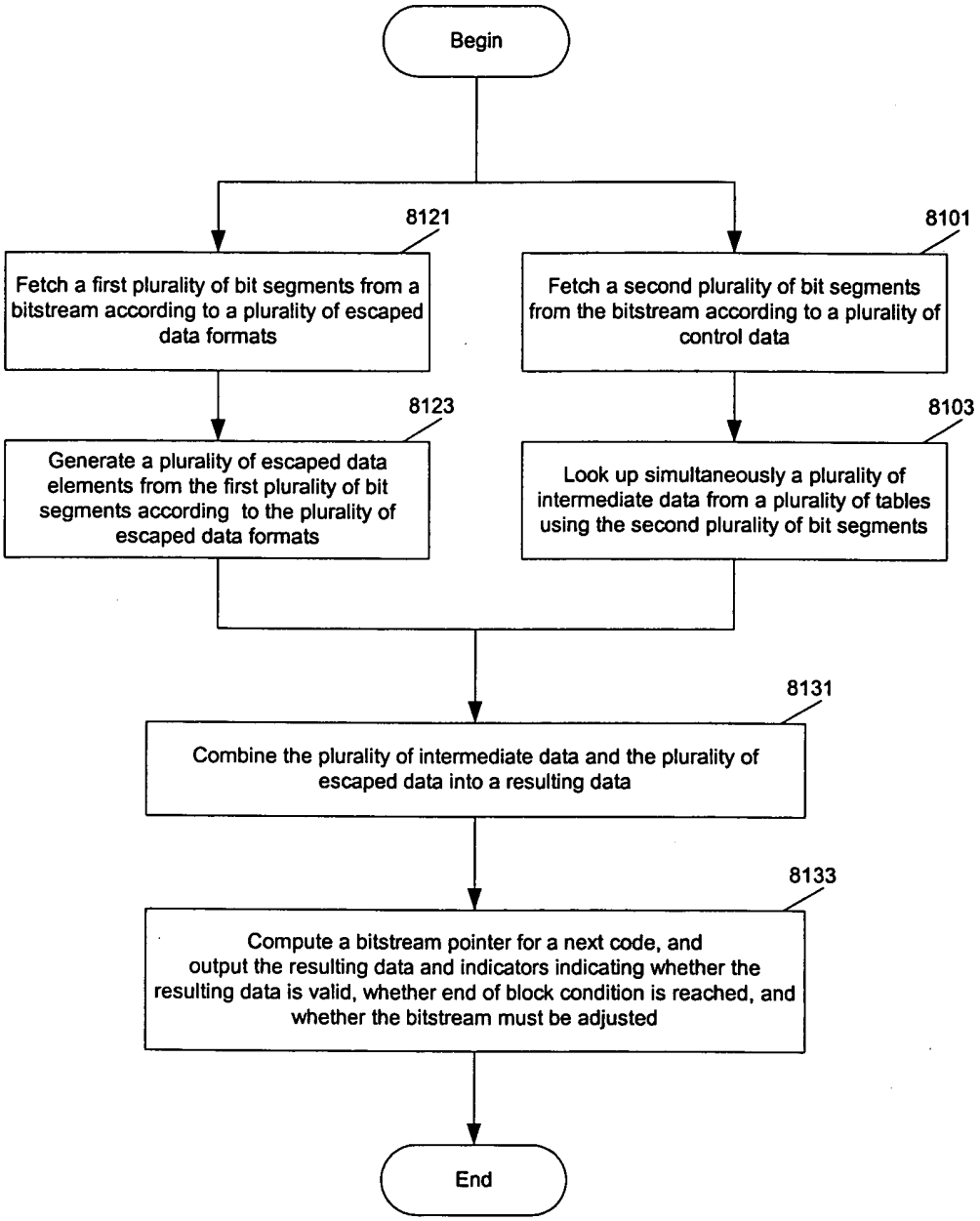

Begin
8121
Fetch a first plurality of bit segments from a bitstream according to a plurality of escaped data formats
8101
Fetch a second plurality of bit segments from the bitstream according to a plurality of control data
8123
Generate a plurality of escaped data elements from the first plurality of bit segments according to the plurality of escaped data formats
8103
Look up simultaneously a plurality of intermediate data from a plurality of tables using the second plurality of bit segments
8131
Combine the plurality of intermediate data and the plurality of escaped data into a resulting data
8133
Compute a bitstream pointer for a next code, and output the resulting data and indicators indicating whether the resulting data is valid, whether end of block condition is reached, and whether the bitstream must be adjusted
End

Fig. 61

Index	T1				T2				T3				T4			
	type	bits	run	level	type	bits	run	level	type	bits	run	level	type	bits	run	level
00	4	2	0	1	4	6	3	1	4	9	11	1	4	10	5	3
01	4	2	0	1	4	6	3	1	4	9	11	-1	4	10	5	3
02	4	2	0	1	4	6	3	-1	4	9	12	1	4	10	5	-3
03	4	2	0	1	4	6	3	-1	4	9	12	-1	4	10	5	-3
04	4	2	0	1	4	6	4	1	4	9	13	1	4	10	3	4
05	4	2	0	1	4	6	4	1	4	9	13	-1	4	10	3	4
06	4	2	0	1	4	6	4	-1	4	9	14	1	4	10	3	-4
07	4	2	0	1	4	6	4	-1	4	9	14	-1	4	10	3	-4
08	4	2	0	-1	4	6	0	7	4	9	5	2	4	10	3	5
09	4	2	0	-1	4	6	0	7	4	9	5	-2	4	10	3	5
0a	4	2	0	-1	4	6	0	-7	4	9	6	2	4	10	3	-5
0b	4	2	0	-1	4	6	0	-7	4	9	6	-2	4	10	3	-5
0c	4	2	0	-1	4	6	0	8	4	9	3	3	4	10	2	6
0d	4	2	0	-1	4	6	0	8	4	9	3	-3	4	10	2	6
0e	4	2	0	-1	4	6	0	-8	4	9	4	3	4	10	2	-6
0f	4	2	0	-1	4	6	0	-8	4	9	4	-3	4	10	2	-6
10	4	3	0	2	4	7	5	1	4	9	2	4	4	10	1	9
11	4	3	0	2	4	7	5	-1	4	9	2	-4	4	10	1	9
12	4	3	0	2	4	7	6	1	4	9	2	5	4	10	1	-9
13	4	3	0	2	4	7	6	-1	4	9	2	-5	4	10	1	-9
14	4	3	0	-2	4	7	2	2	4	9	1	8	4	10	1	10
15	4	3	0	-2	4	7	2	-2	4	9	1	-8	4	10	1	10
16	4	3	0	-2	4	7	1	3	4	9	0	18	4	10	1	-10
17	4	3	0	-2	4	7	1	-3	4	9	0	-18	4	10	1	-10
18	1	3	0	0	4	7	1	4	4	9	0	19	4	10	1	11
19	1	3	0	0	4	7	1	-4	4	9	0	-19	4	10	1	11
1a	1	3	0	0	4	7	0	9	4	9	0	20	4	10	1	-11
1b	1	3	0	0	4	7	0	-9	4	9	0	-20	4	10	1	-11
1c	4	4	1	1	4	7	0	10	4	9	0	21	4	10	0	0
1d	4	4	1	1	4	7	0	-10	4	9	0	-21	4	10	0	0
1e	4	4	1	-1	4	7	0	11	4	9	0	22	4	10	1	0
1f	4	4	1	-1	4	7	0	-11	4	9	0	-22	4	10	1	0
20	4	4	0	3	5	8	7	1	0	0	0	0	4	11	6	3
21	4	4	0	3	5	8	8	1	0	0	0	0	4	11	6	-3
22	4	4	0	-3	5	8	9	1	0	0	0	0	4	11	4	4
23	4	4	0	-3	5	8	10	1	0	0	0	0	4	11	4	-4
24	4	4	0	4	5	8	3	2	0	0	0	0	4	11	3	6
25	4	4	0	4	5	8	4	2	0	0	0	0	4	11	3	-6
26	4	4	0	-4	5	8	2	3	0	0	0	0	4	11	1	12
27	4	4	0	-4	5	8	1	5	0	0	0	0	4	11	1	-12
28	4	5	2	1	5	8	1	6	0	0	0	0	4	11	1	13
29	4	5	2	-1	5	8	1	7	0	0	0	0	4	11	1	-13
2a	4	5	1	2	5	8	0	12	0	0	0	0	4	11	1	14
2b	4	5	1	-2	5	8	0	13	0	0	0	0	4	11	1	-14
2c	4	5	0	5	5	8	0	14	0	0	0	0	4	11	2	0
2d	4	5	0	-5	5	8	0	15	0	0	0	0	4	11	3	0
2e	4	5	0	6	5	8	0	16	0	0	0	0	4	11	4	0
2f	4	5	0	-6	5	8	0	17	0	0	0	0	4	11	5	0
30	0	0	0	0	0	0	0	0	0	0	0	0	5	12	7	2
31	0	0	0	0	0	0	0	0	0	0	0	0	5	12	8	2
32	0	0	0	0	0	0	0	0	0	0	0	0	5	12	9	2
33	0	0	0	0	0	0	0	0	0	0	0	0	5	12	10	2
34	0	0	0	0	0	0	0	0	0	0	0	0	5	12	7	3
35	0	0	0	0	0	0	0	0	0	0	0	0	5	12	8	3
36	0	0	0	0	0	0	0	0	0	0	0	0	5	12	4	5
37	0	0	0	0	0	0	0	0	0	0	0	0	5	12	3	7
38	0	0	0	0	0	0	0	0	0	0	0	0	5	12	2	7
39	0	0	0	0	0	0	0	0	0	0	0	0	5	12	2	8
3a	0	0	0	0	0	0	0	0	0	0	0	0	5	12	2	9
3b	0	0	0	0	0	0	0	0	0	0	0	0	5	12	2	10
3c	0	0	0	0	2	12	0	0	0	0	0	0	5	12	2	11
3d	0	0	0	0	2	12	0	0	0	0	0	0	5	12	1	15
3e	0	0	0	0	3	15	0	0	0	0	0	0	5	12	1	16
3f	0	0	0	0	3	15	0	0	0	0	0	0	5	12	1	17
40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ff	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 62

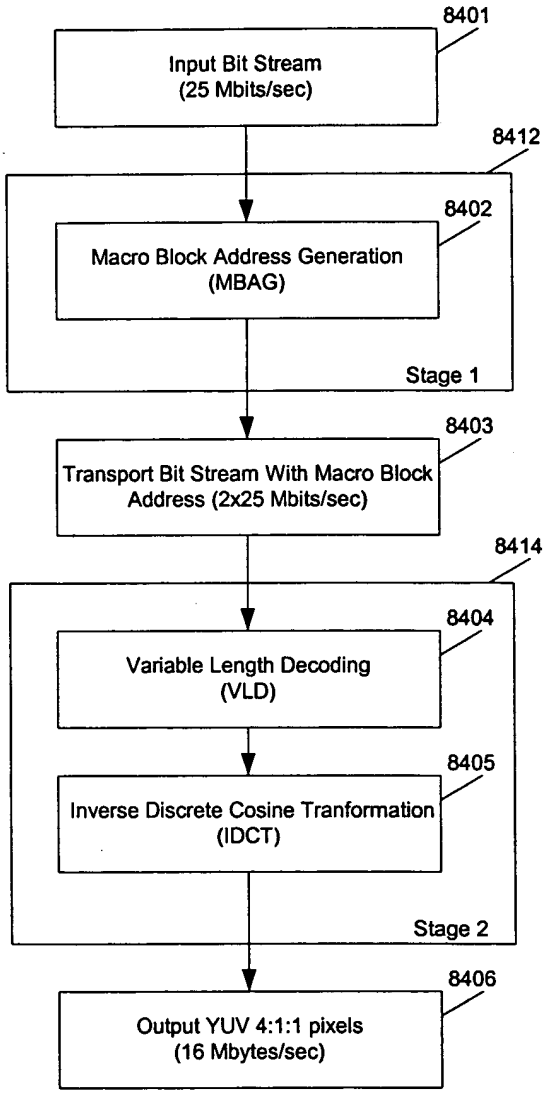

8401
Input Bit Stream
(25 Mbits/sec)
8412
8402
Macro Block Address Generation
(MBAG)
Stage 1
8403
Transport Bit Stream With Macro Block
Address (2x25 Mbits/sec)
8414
8404
Variable Length Decoding
(VLD)
8405
Inverse Discrete Cosine Tranformation
(IDCT)
Stage 2
8406
Output YUV 4:1:1 pixels
(16 Mbytes/sec)

Fig. 64

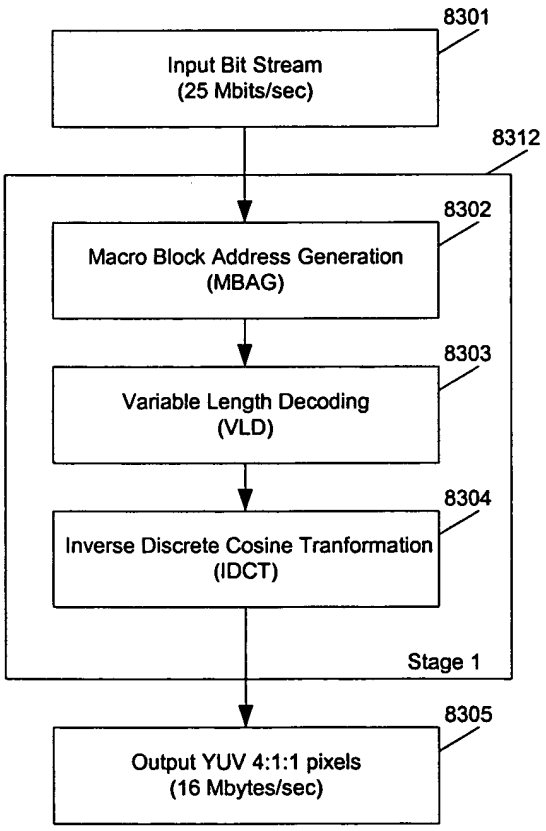

8301
Input Bit Stream
(25 Mbits/sec)
8312
8302
Macro Block Address Generation
(MBAG)
8303
Variable Length Decoding
(VLD)
8304
Inverse Discrete Cosine Tranformation
(IDCT)
Stage 1
8305
Output YUV 4:1:1 pixels
(16 Mbytes/sec)

Fig. 63

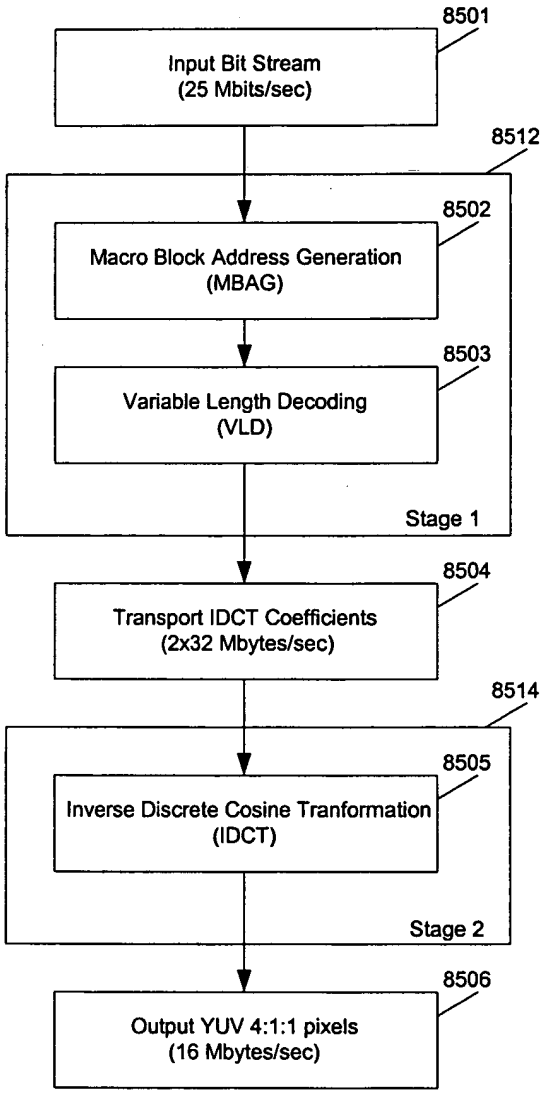

8501
Input Bit Stream
(25 Mbits/sec)
8512
8502
Macro Block Address Generation
(MBAG)
8503
Variable Length Decoding
(VLD)
Stage 1
8504
Transport IDCT Coefficients
(2x32 Mbytes/sec)
8514
8505
Inverse Discrete Cosine Tranformation
(IDCT)
Stage 2
8506
Output YUV 4:1:1 pixels
(16 Mbytes/sec)

Fig. 65

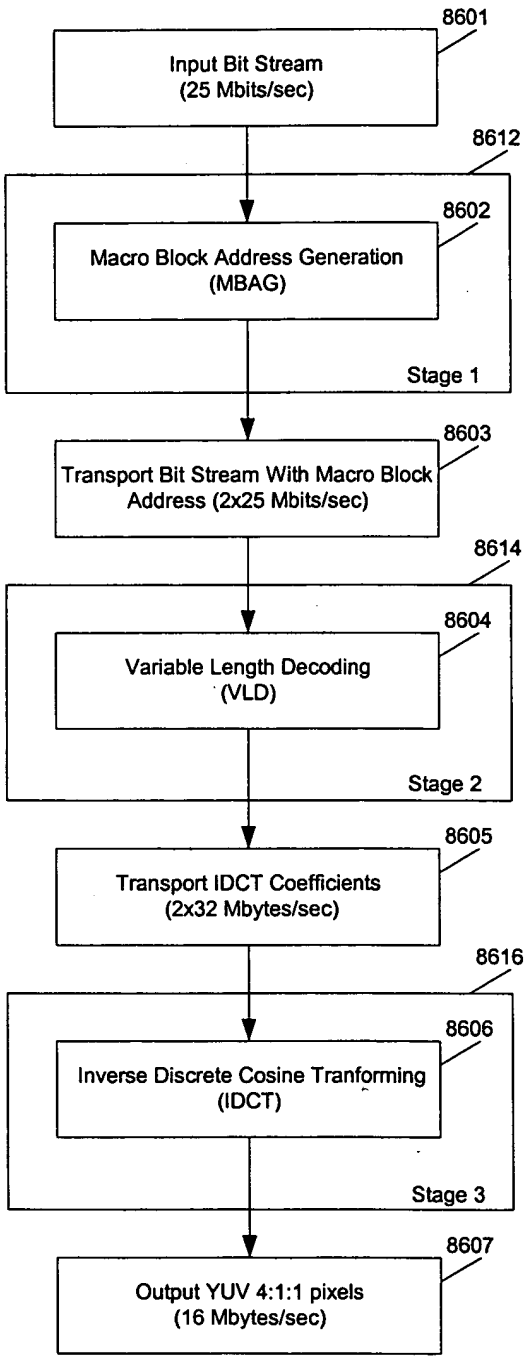

8601
Input Bit Stream
(25 Mbits/sec)
8612
8602
Macro Block Address Generation
(MBAG)
Stage 1
8603
Transport Bit Stream With Macro Block
Address (2x25 Mbits/sec)
8614
8604
Variable Length Decoding
(VLD)
Stage 2
8605
Transport IDCT Coefficients
(2x32 Mbytes/sec)
8616
8606
Inverse Discrete Cosine Tranforming
(IDCT)
Stage 3
8607
Output YUV 4:1:1 pixels
(16 Mbytes/sec)

Fig. 66

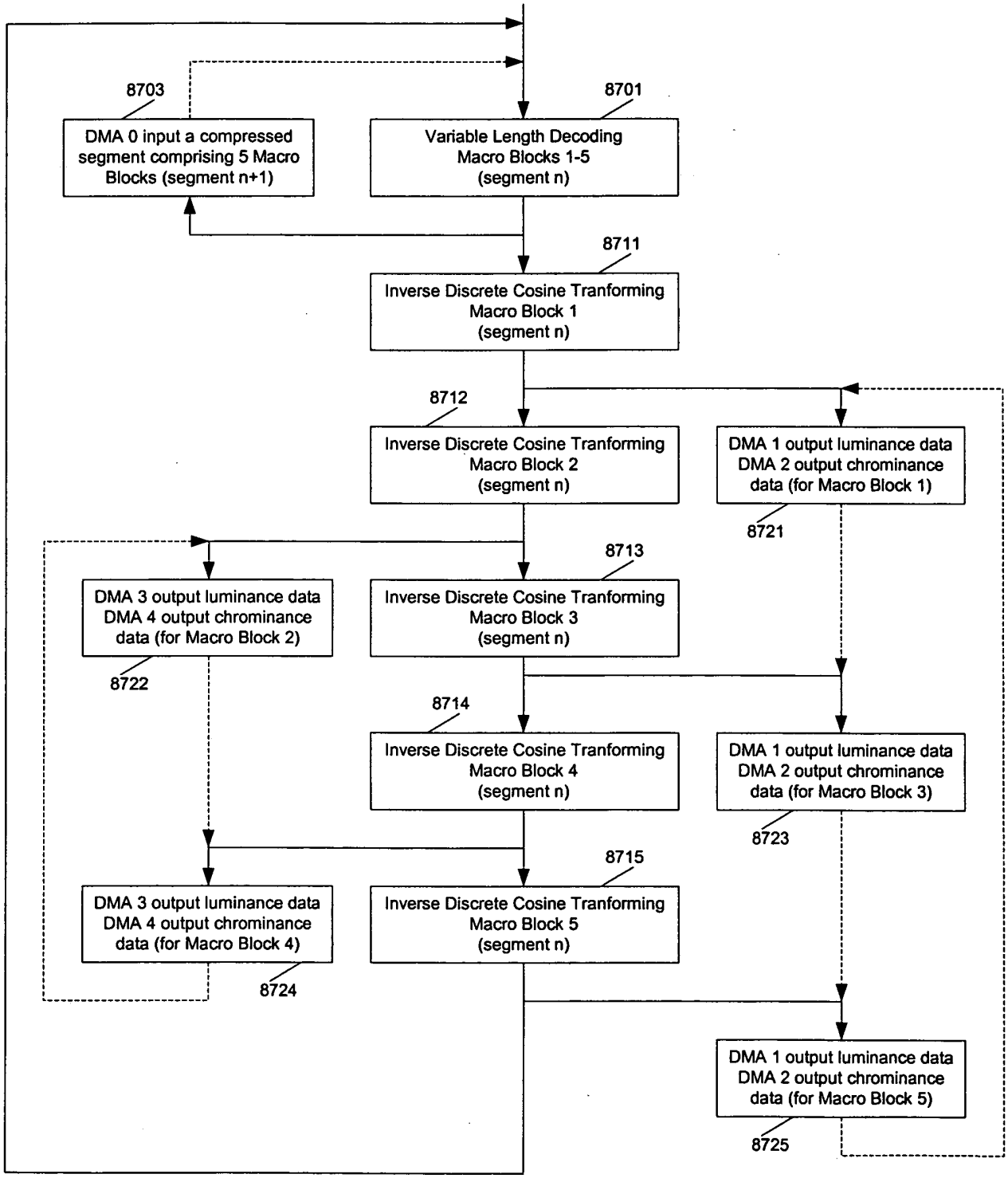

8703
DMA 0 input a compressed segment comprising 5 Macro Blocks (segment n+1)
8701
Variable Length Decoding Macro Blocks 1-5 (segment n)
8711
Inverse Discrete Cosine Tranforming Macro Block 1 (segment n)
8712
Inverse Discrete Cosine Tranforming Macro Block 2 (segment n)
DMA 1 output luminance data DMA 2 output chrominance data (for Macro Block 1)
8721
8713
DMA 3 output luminance data DMA 4 output chrominance data (for Macro Block 2)
8722
Inverse Discrete Cosine Tranforming Macro Block 3 (segment n)
8714
Inverse Discrete Cosine Tranforming Macro Block 4 (segment n)
DMA 1 output luminance data DMA 2 output chrominance data (for Macro Block 3)
8723
8715
DMA 3 output luminance data DMA 4 output chrominance data (for Macro Block 4)
8724
Inverse Discrete Cosine Tranforming Macro Block 5 (segment n)
DMA 1 output luminance data DMA 2 output chrominance data (for Macro Block 5)
8725

Fig. 67

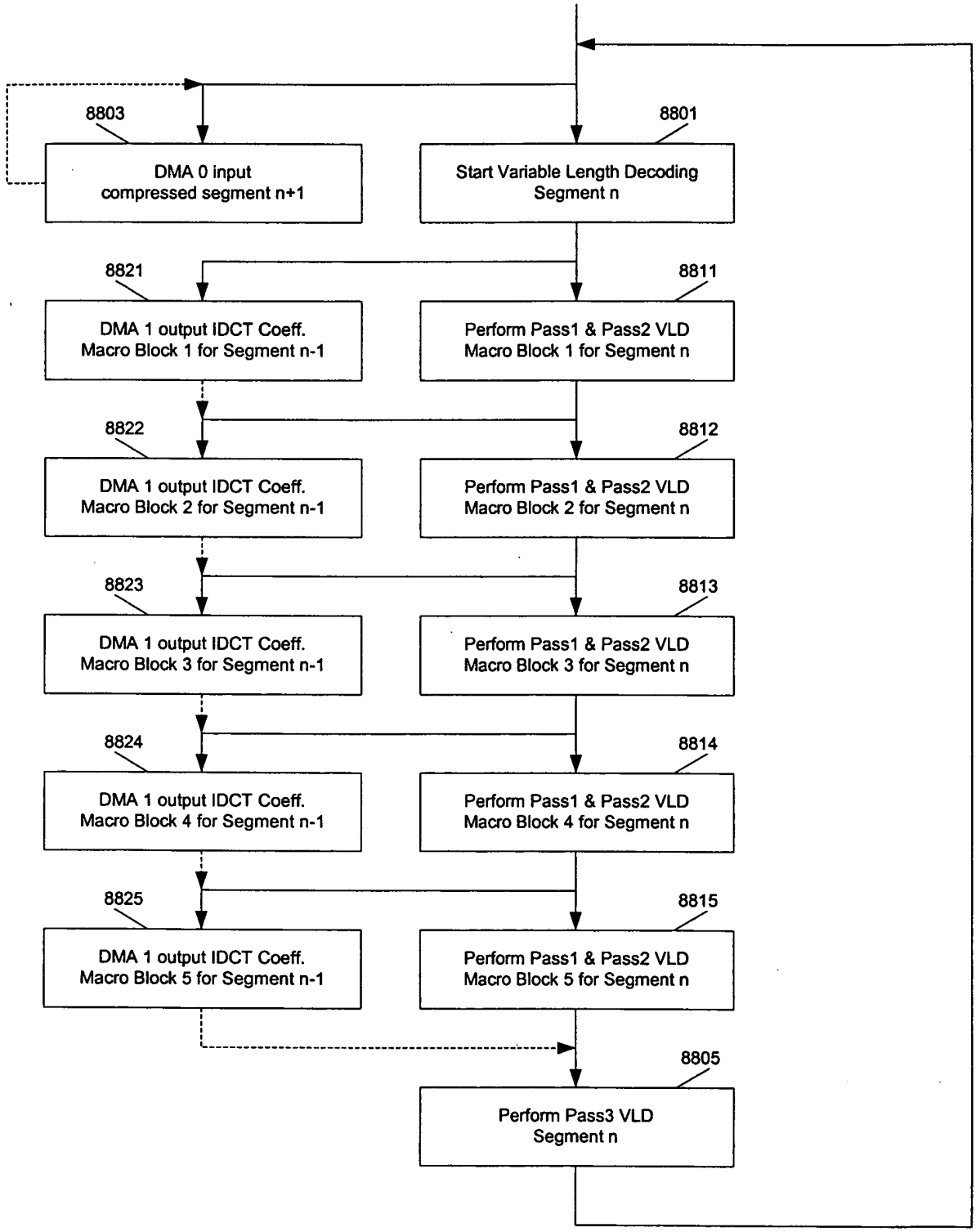

8803
DMA 0 input
compressed segment n+1
8801
Start Variable Length Decoding
Segment n
8821
DMA 1 output IDCT Coeff.
Macro Block 1 for Segment n-1
8811
Perform Pass1 & Pass2 VLD
Macro Block 1 for Segment n
8822
DMA 1 output IDCT Coeff.
Macro Block 2 for Segment n-1
8812
Perform Pass1 & Pass2 VLD
Macro Block 2 for Segment n
8823
DMA 1 output IDCT Coeff.
Macro Block 3 for Segment n-1
8813
Perform Pass1 & Pass2 VLD
Macro Block 3 for Segment n
8824
DMA 1 output IDCT Coeff.
Macro Block 4 for Segment n-1
8814
Perform Pass1 & Pass2 VLD
Macro Block 4 for Segment n
8825
DMA 1 output IDCT Coeff.
Macro Block 5 for Segment n-1
8815
Perform Pass1 & Pass2 VLD
Macro Block 5 for Segment n
8805
Perform Pass3 VLD
Segment n

Fig. 68

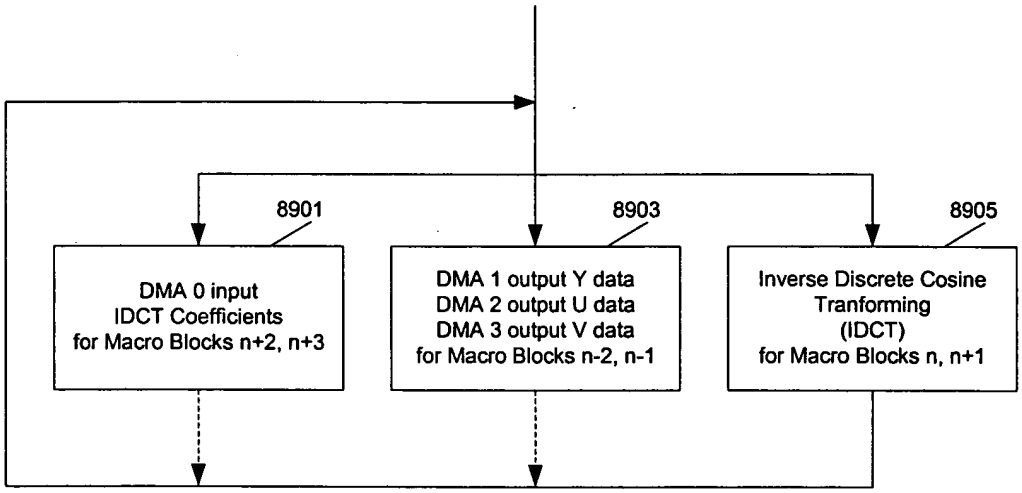

8901
8903
8905
DMA 0 input
IDCT Coefficients
for Macro Blocks n+2, n+3
DMA 1 output Y data
DMA 2 output U data
DMA 3 output V data
for Macro Blocks n-2, n-1
Inverse Discrete Cosine
Tranforming
(IDCT)
for Macro Blocks n, n+1

Fig. 69

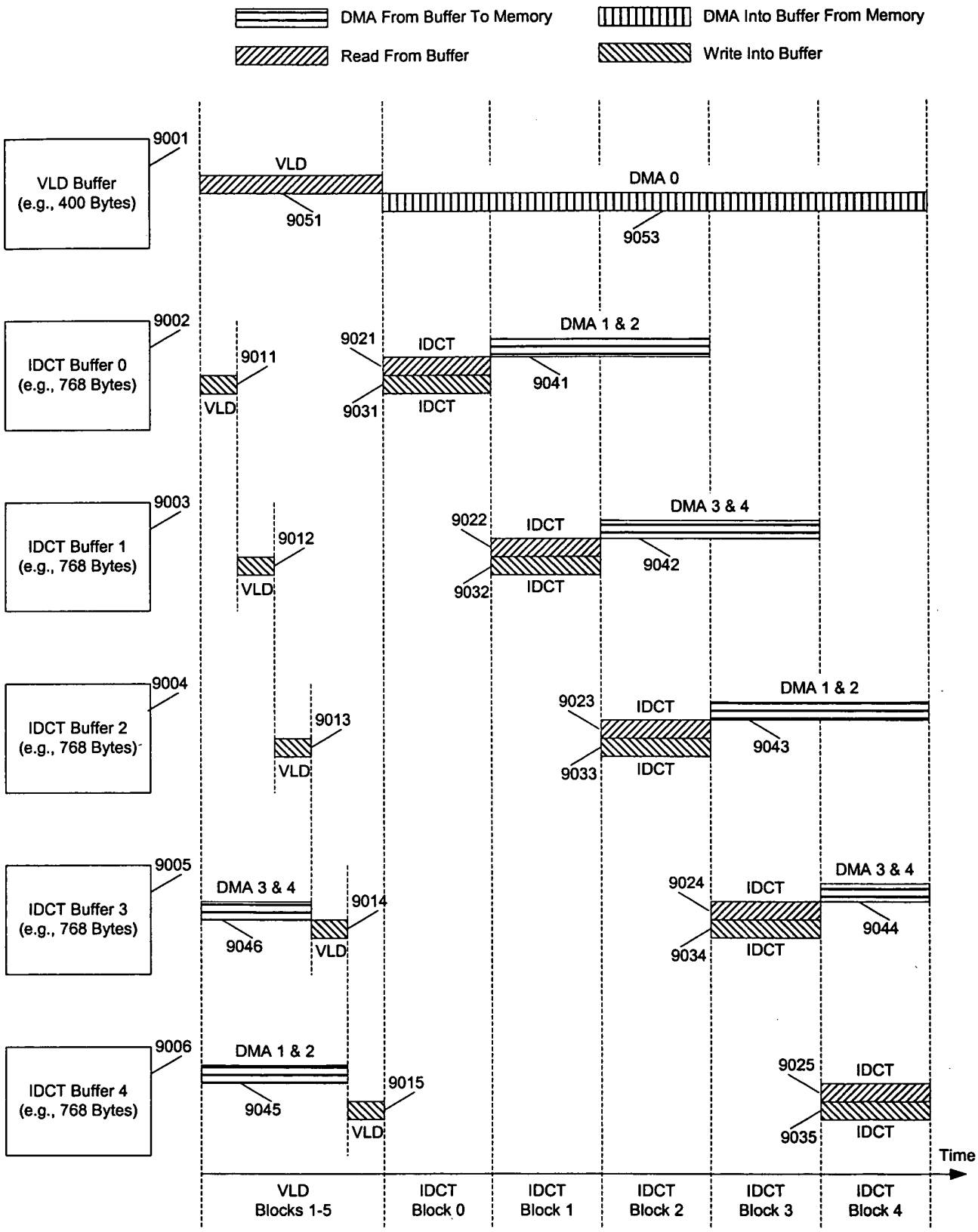

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
9001
VLD Buffer
(e.g., 400 Bytes)
VLD
9051
DMA 0
9053
9002
IDCT Buffer 0
(e.g., 768 Bytes)
9011
VLD
9021
IDCT
9031
IDCT
DMA 1 & 2
9041
9003
IDCT Buffer 1
(e.g., 768 Bytes)
9012
VLD
9022
IDCT
9032
IDCT
DMA 3 & 4
9042
9004
IDCT Buffer 2
(e.g., 768 Bytes)
9013
VLD
9023
IDCT
9033
IDCT
DMA 1 & 2
9043
9005
IDCT Buffer 3
(e.g., 768 Bytes)
DMA 3 & 4
9046
9014
VLD
9024
IDCT
9034
IDCT
DMA 3 & 4
9044
9006
IDCT Buffer 4
(e.g., 768 Bytes)
DMA 1 & 2
9045
9015
VLD
9025
IDCT
9035
IDCT
Time
VLD
Blocks 1-5
IDCT
Block 0
IDCT
Block 1
IDCT
Block 2
IDCT
Block 3
IDCT
Block 4

Fig. 70

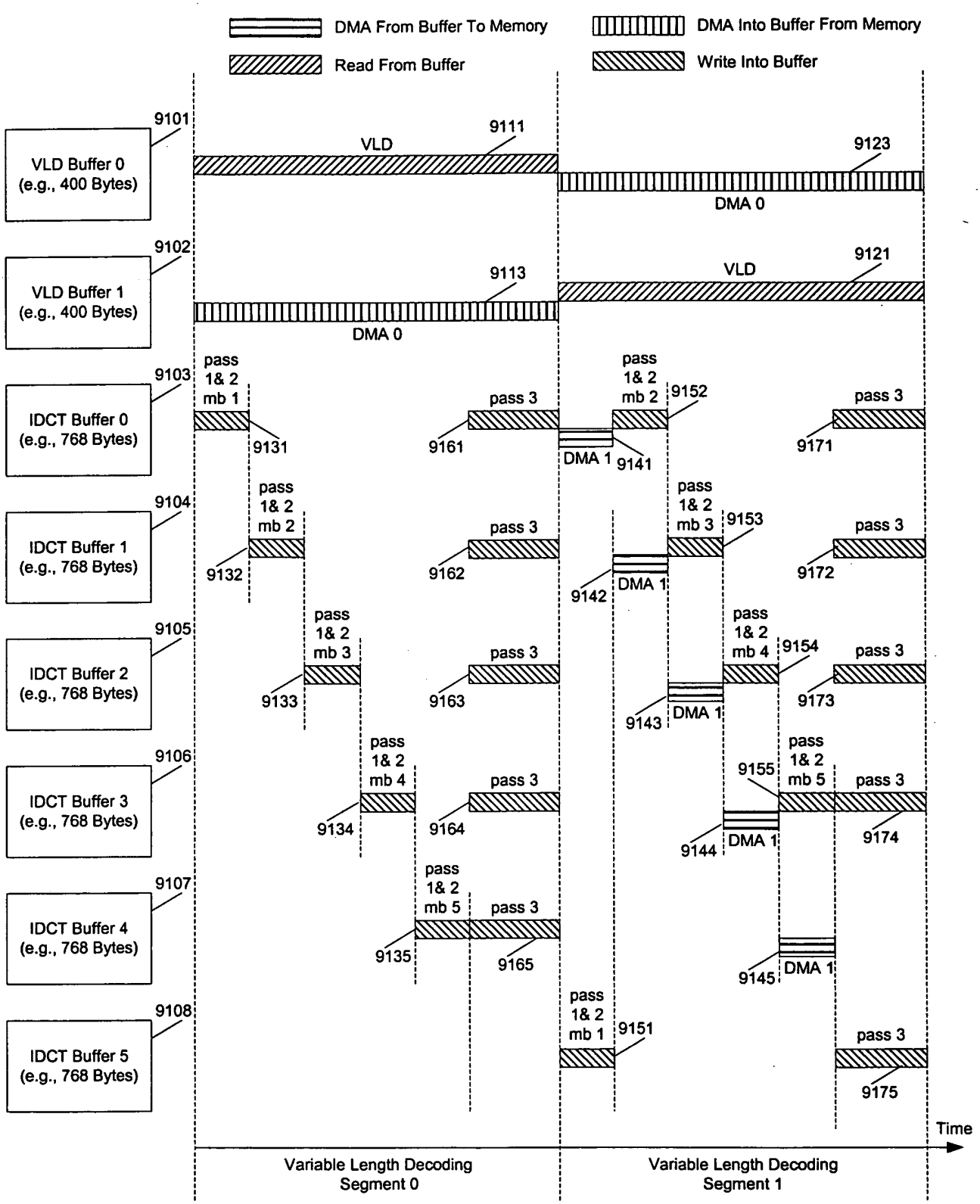

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
VLD Buffer 0 (e.g., 400 Bytes)
VLD Buffer 1 (e.g., 400 Bytes)
IDCT Buffer 0 (e.g., 768 Bytes)
IDCT Buffer 1 (e.g., 768 Bytes)
IDCT Buffer 2 (e.g., 768 Bytes)
IDCT Buffer 3 (e.g., 768 Bytes)
IDCT Buffer 4 (e.g., 768 Bytes)
IDCT Buffer 5 (e.g., 768 Bytes)
9101
9102
9103
9104
9105
9106
9107
9108
VLD
DMA 0
DMA 1
pass 1& 2 mb 1
pass 1& 2 mb 2
pass 1& 2 mb 3
pass 1& 2 mb 4
pass 1& 2 mb 5
pass 3
9111
9113
9121
9123
9131
9132
9133
9134
9135
9141
9142
9143
9144
9145
9151
9152
9153
9154
9155
9161
9162
9163
9164
9165
9171
9172
9173
9174
9175
Time
Variable Length Decoding Segment 0
Variable Length Decoding Segment 1

Fig. 71

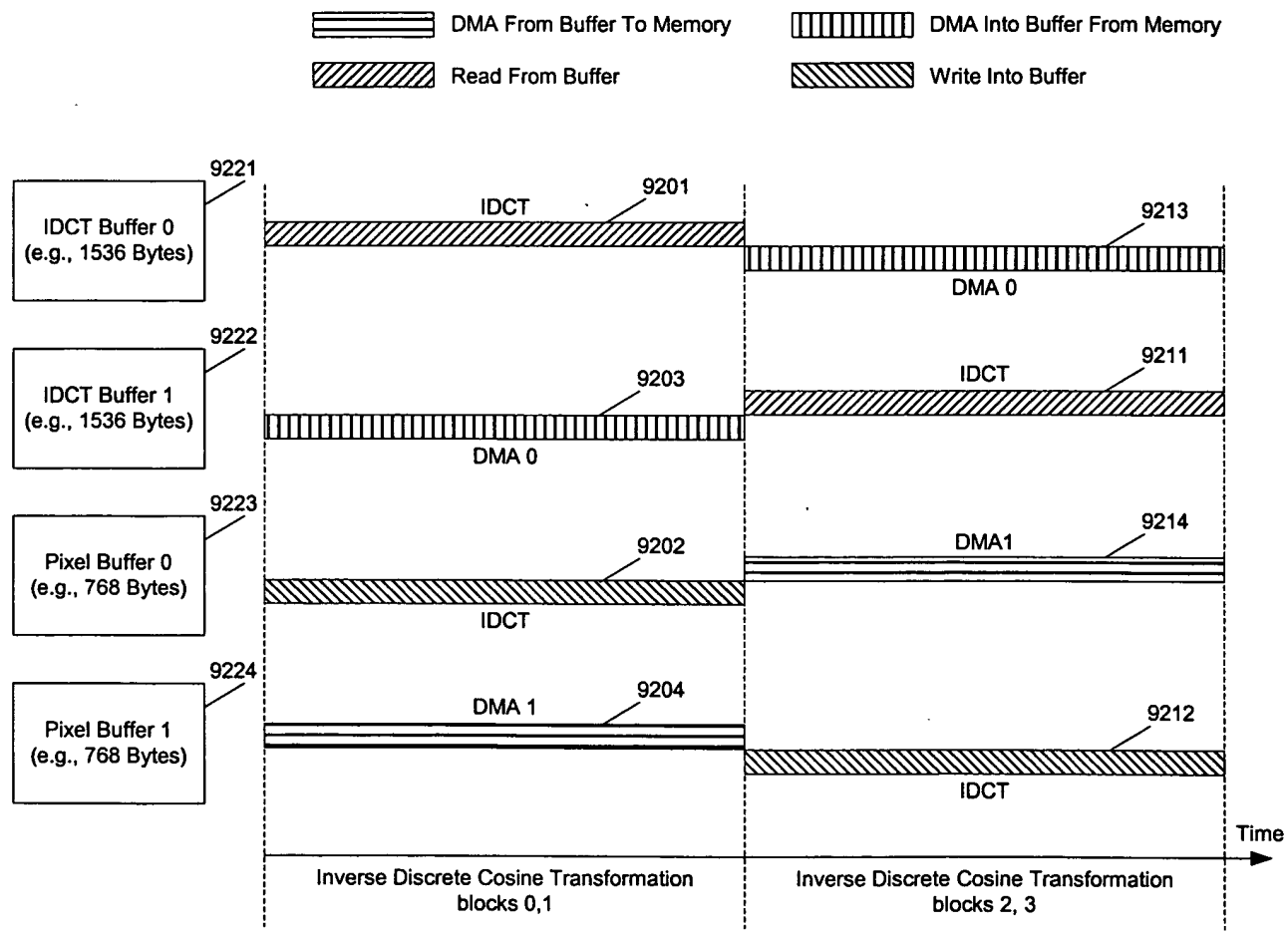

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
9221
IDCT Buffer 0
(e.g., 1536 Bytes)
9222
IDCT Buffer 1
(e.g., 1536 Bytes)
9223
Pixel Buffer 0
(e.g., 768 Bytes)
9224
Pixel Buffer 1
(e.g., 768 Bytes)
IDCT
9201
9213
DMA 0
9203
DMA 0
IDCT
9211
9202
IDCT
DMA1
9214
DMA 1
9204
IDCT
9212
Time
Inverse Discrete Cosine Transformation
blocks 0,1
Inverse Discrete Cosine Transformation
blocks 2, 3

Fig. 72

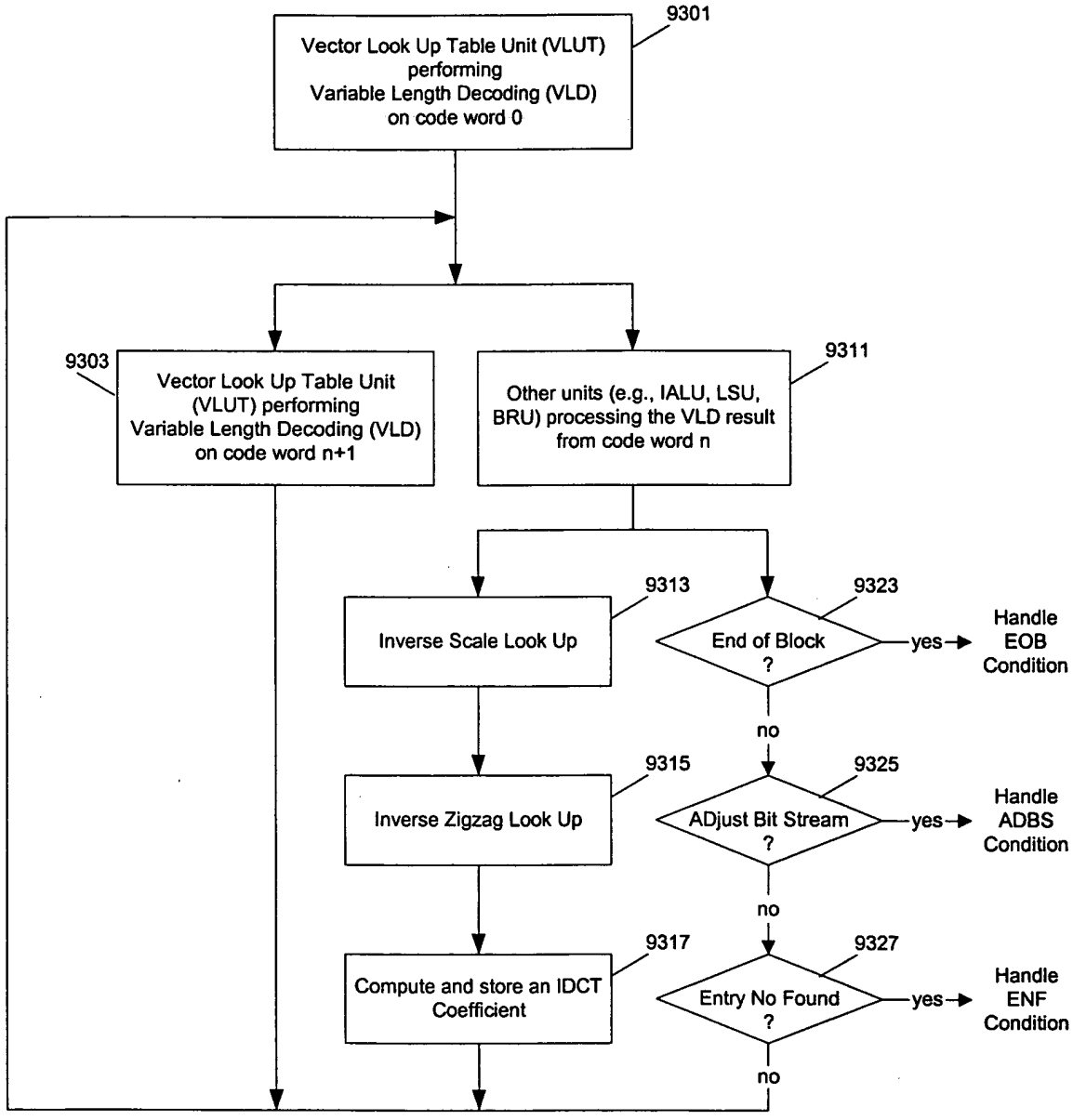
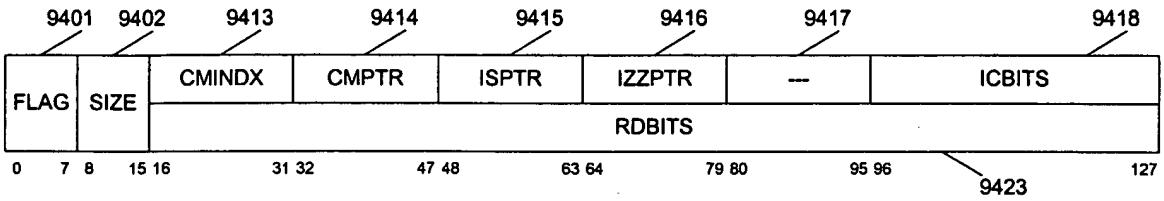

9301
Vector Look Up Table Unit (VLUT) performing Variable Length Decoding (VLD) on code word 0
9303
Vector Look Up Table Unit (VLUT) performing Variable Length Decoding (VLD) on code word n+1
9311
Other units (e.g., IALU, LSU, BRU) processing the VLD result from code word n
9313
Inverse Scale Look Up
9315
Inverse Zigzag Look Up
9317
Compute and store an IDCT Coefficient
9323
End of Block ?
yes
Handle EOB Condition
no
9325
ADjust Bit Stream ?
yes
Handle ADBS Condition
no
9327
Entry No Found ?
yes
Handle ENF Condition
no

Fig. 73


9401
9402
9413
9414
9415
9416
9417
9418
FLAG
SIZE
CMINDX
CMPTR
ISPTR
IZZPTR

ICBITS
RDBITS
0 7 8 15 16 31 32 47 48 63 64 79 80 95 96 127
9423

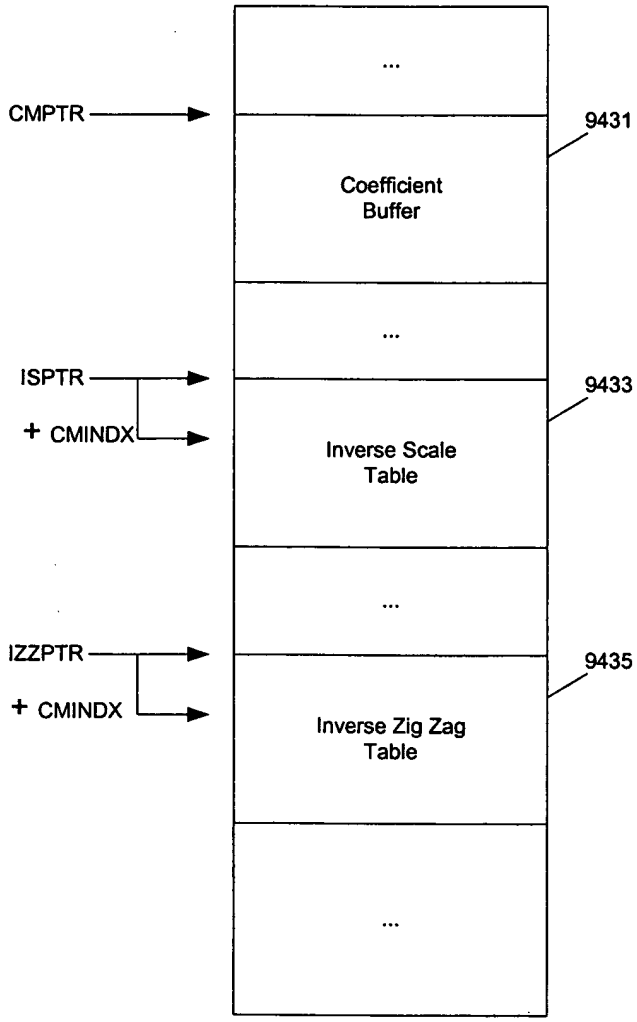

CMPTR
ISPTR
+ CMINDX
IZZPTR
+ CMINDX
...
Coefficient Buffer
Inverse Scale Table
Inverse Zig Zag Table
9431
9433
9435

Fig. 74

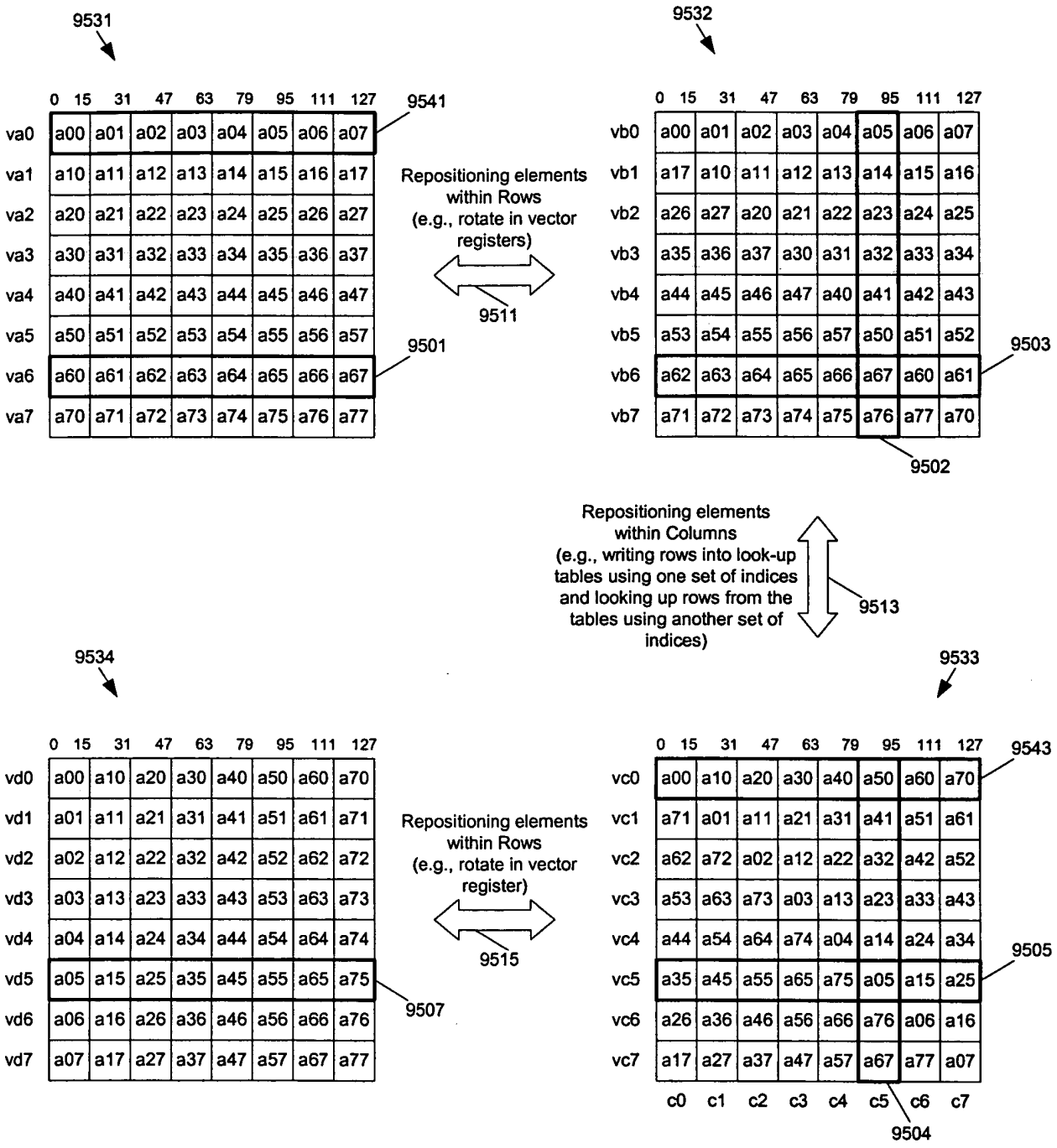

9531
9532
0 15 31 47 63 79 95 111 127
9541
va0 a00 a01 a02 a03 a04 a05 a06 a07
va1 a10 a11 a12 a13 a14 a15 a16 a17
va2 a20 a21 a22 a23 a24 a25 a26 a27
va3 a30 a31 a32 a33 a34 a35 a36 a37
va4 a40 a41 a42 a43 a44 a45 a46 a47
va5 a50 a51 a52 a53 a54 a55 a56 a57
va6 a60 a61 a62 a63 a64 a65 a66 a67
va7 a70 a71 a72 a73 a74 a75 a76 a77
9501
Repositioning elements within Rows (e.g., rotate in vector registers)
9511
vb0 a00 a01 a02 a03 a04 a05 a06 a07
vb1 a17 a10 a11 a12 a13 a14 a15 a16
vb2 a26 a27 a20 a21 a22 a23 a24 a25
vb3 a35 a36 a37 a30 a31 a32 a33 a34
vb4 a44 a45 a46 a47 a40 a41 a42 a43
vb5 a53 a54 a55 a56 a57 a50 a51 a52
vb6 a62 a63 a64 a65 a66 a67 a60 a61
vb7 a71 a72 a73 a74 a75 a76 a77 a70
9503
9502
Repositioning elements within Columns (e.g., writing rows into look-up tables using one set of indices and looking up rows from the tables using another set of indices)
9513
9534
9533
0 15 31 47 63 79 95 111 127
vd0 a00 a10 a20 a30 a40 a50 a60 a70
vd1 a01 a11 a21 a31 a41 a51 a61 a71
vd2 a02 a12 a22 a32 a42 a52 a62 a72
vd3 a03 a13 a23 a33 a43 a53 a63 a73
vd4 a04 a14 a24 a34 a44 a54 a64 a74
vd5 a05 a15 a25 a35 a45 a55 a65 a75
vd6 a06 a16 a26 a36 a46 a56 a66 a76
vd7 a07 a17 a27 a37 a47 a57 a67 a77
9507
Repositioning elements within Rows (e.g., rotate in vector register)
9515
9543
vc0 a00 a10 a20 a30 a40 a50 a60 a70
vc1 a71 a01 a11 a21 a31 a41 a51 a61
vc2 a62 a72 a02 a12 a22 a32 a42 a52
vc3 a53 a63 a73 a03 a13 a23 a33 a43
vc4 a44 a54 a64 a74 a04 a14 a24 a34
vc5 a35 a45 a55 a65 a75 a05 a15 a25
vc6 a26 a36 a46 a56 a66 a76 a06 a16
vc7 a17 a27 a37 a47 a57 a67 a77 a07
9505
c0 c1 c2 c3 c4 c5 c6 c7
9504

Fig. 75

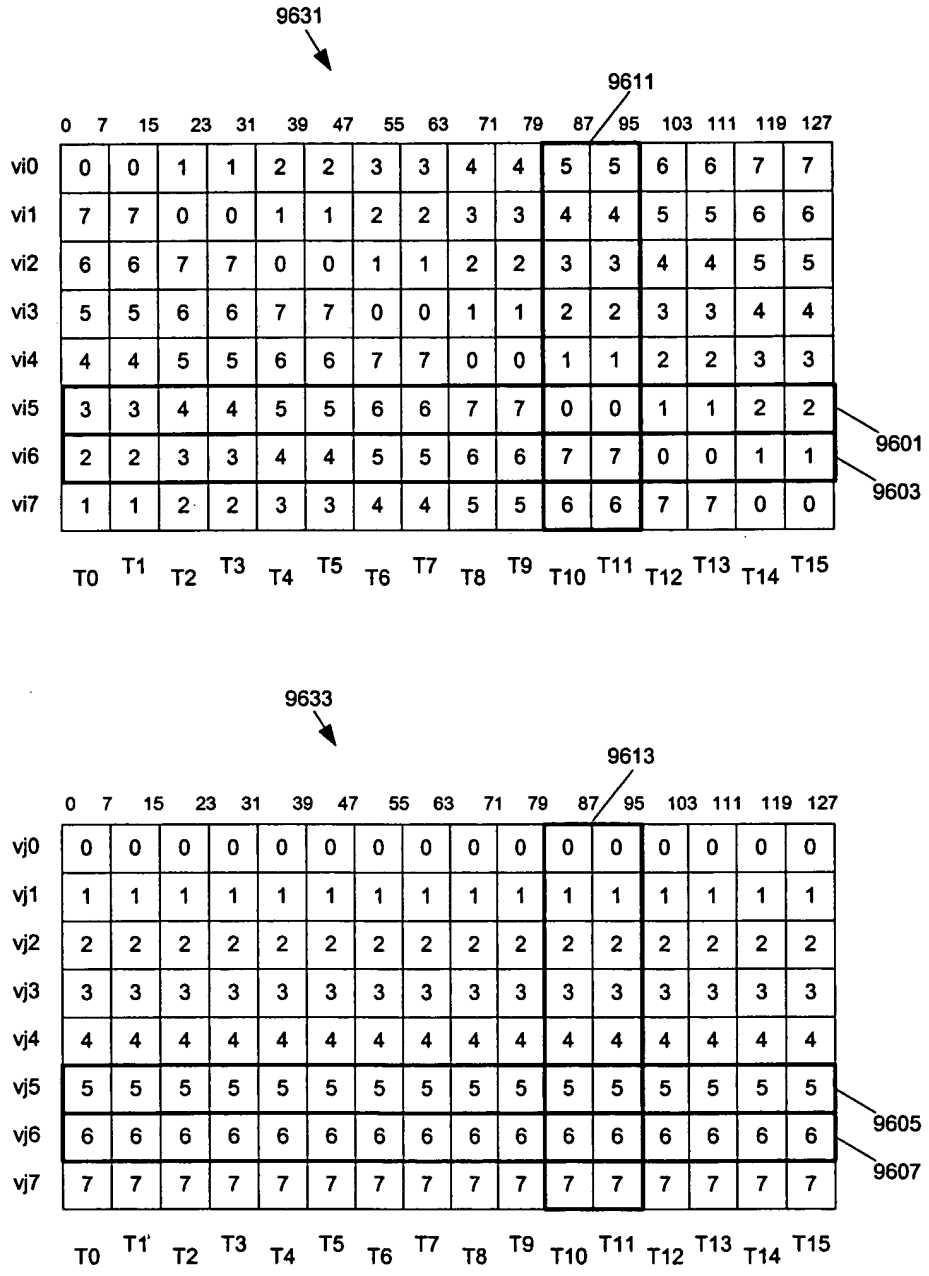

9631
9611
0 7 15 23 31 39 47 55 63 71 79 87 95 103 111 119 127
vi0 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7
vi1 7 7 0 0 1 1 2 2 3 3 4 4 5 5 6 6
vi2 6 6 7 7 0 0 1 1 2 2 3 3 4 4 5 5
vi3 5 5 6 6 7 7 0 0 1 1 2 2 3 3 4 4
vi4 4 4 5 5 6 6 7 7 0 0 1 1 2 2 3 3
vi5 3 3 4 4 5 5 6 6 7 7 0 0 1 1 2 2
vi6 2 2 3 3 4 4 5 5 6 6 7 7 0 0 1 1
vi7 1 1 2 2 3 3 4 4 5 5 6 6 7 7 0 0
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
9601
9603
9633
9613
0 7 15 23 31 39 47 55 63 71 79 87 95 103 111 119 127
vj0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
vj1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
vj2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
vj3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
vj4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
vj5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
vj6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
vj7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
9605
9607

Fig. 76

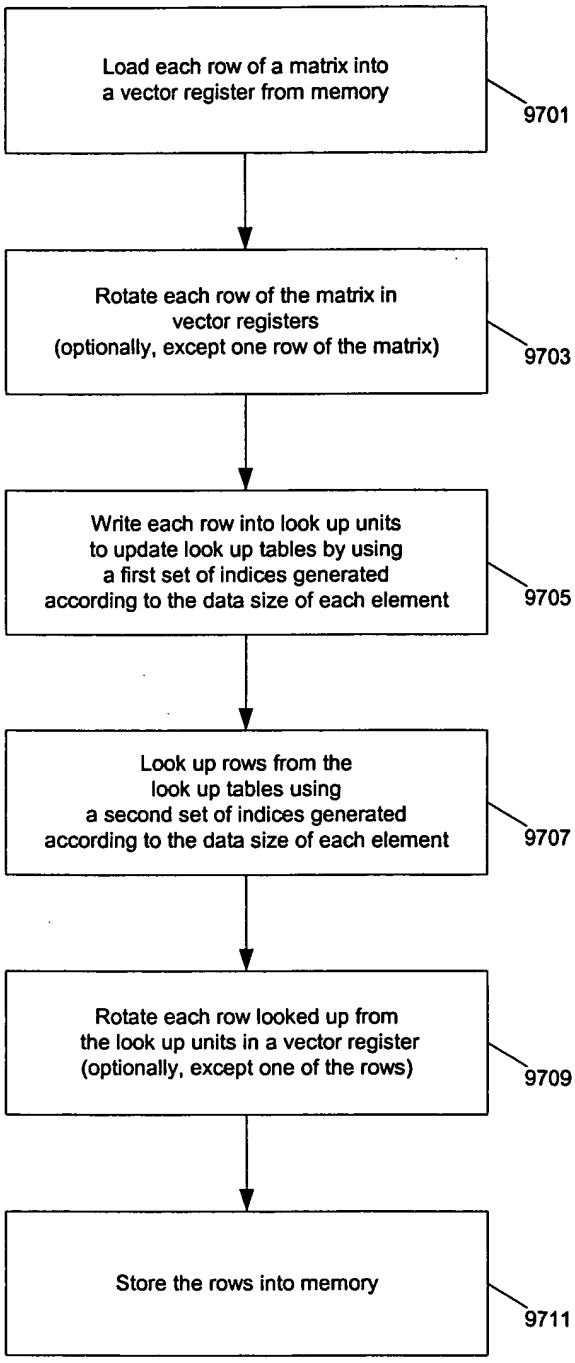

Load each row of a matrix into a vector register from memory
9701
Rotate each row of the matrix in vector registers (optionally, except one row of the matrix)
9703
Write each row into look up units to update look up tables by using a first set of indices generated according to the data size of each element
9705
Look up rows from the look up tables using a second set of indices generated according to the data size of each element
9707
Rotate each row looked up from the look up units in a vector register (optionally, except one of the rows)
9709
Store the rows into memory
9711

Fig. 77

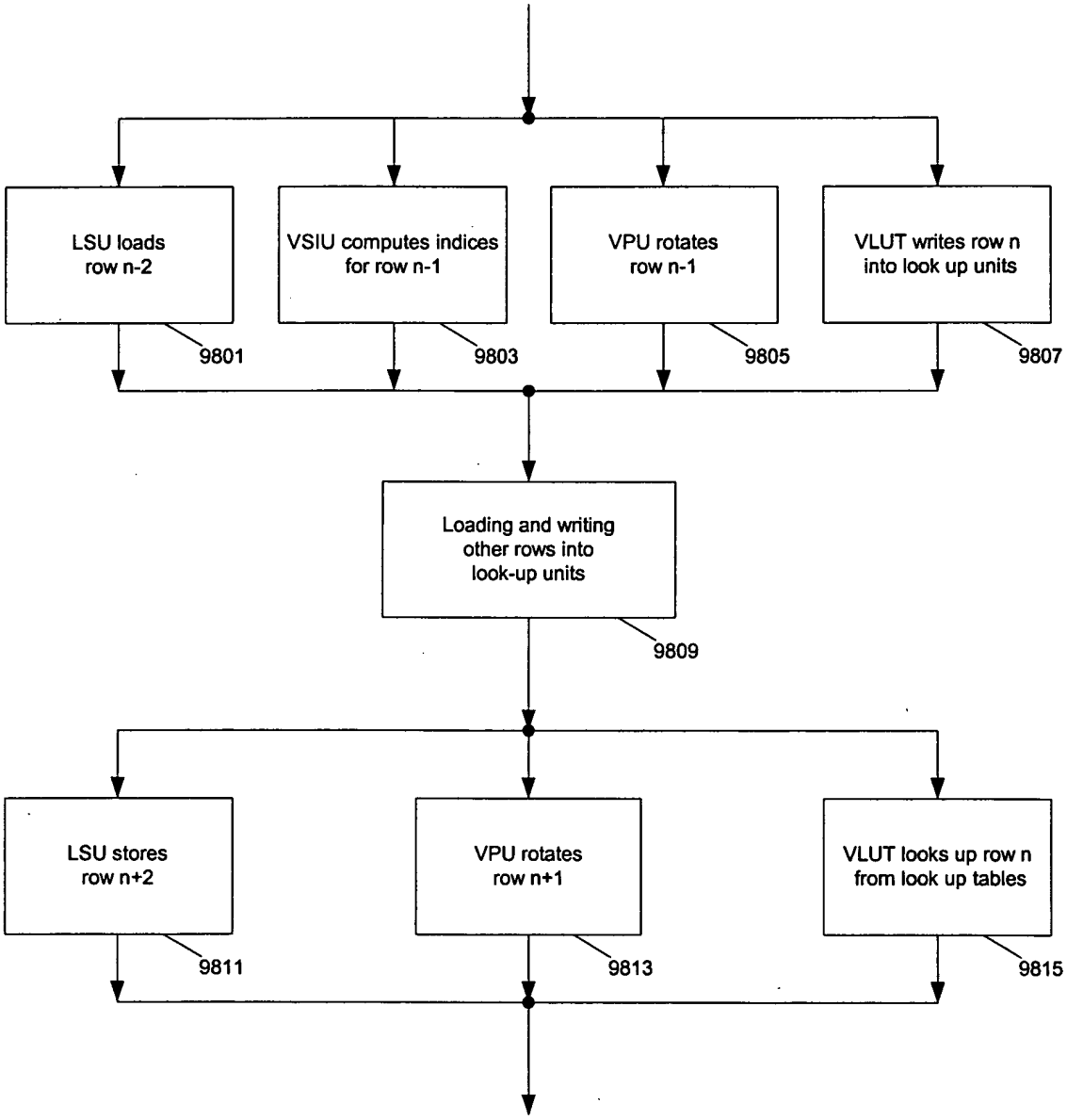

LSU loads row n-2
VSIU computes indices for row n-1
VPU rotates row n-1
VLUT writes row n into look up units
9801
9803
9805
9807
Loading and writing other rows into look-up units
9809
LSU stores row n+2
VPU rotates row n+1
VLUT looks up row n from look up tables
9811
9813
9815

Fig. 78

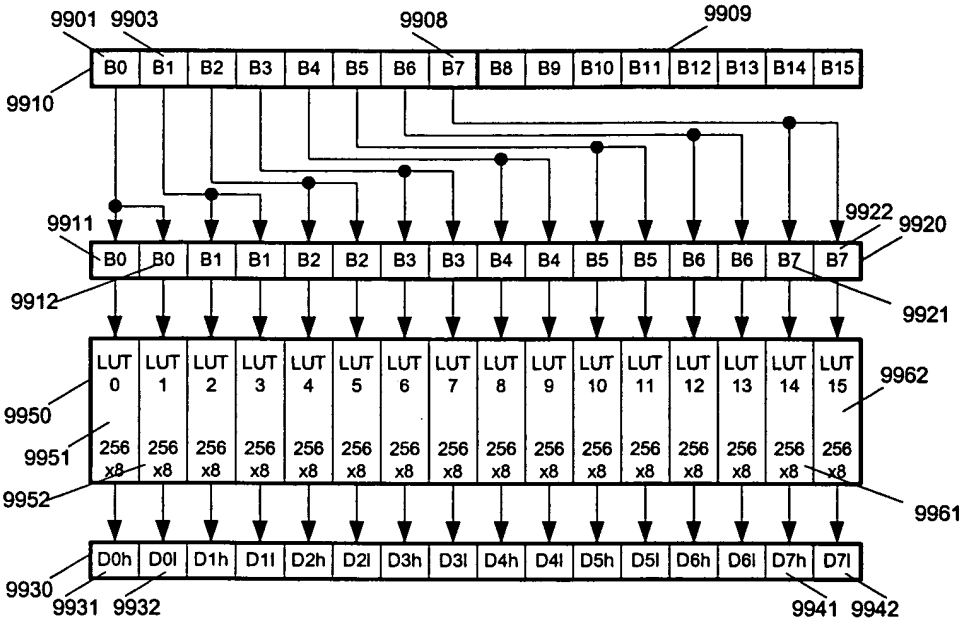

9901
9903
9908
9909
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
9910
9911
9922
9920
B0 B0 B1 B1 B2 B2 B3 B3 B4 B4 B5 B5 B6 B6 B7 B7
9912
9921
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
9962
9950
9951
256 x8
9952
9961
D0h D0l D1h D1l D2h D2l D3h D3l D4h D4l D5h D5l D6h D6l D7h D7l
9930
9931
9932
9941
9942

Fig. 79

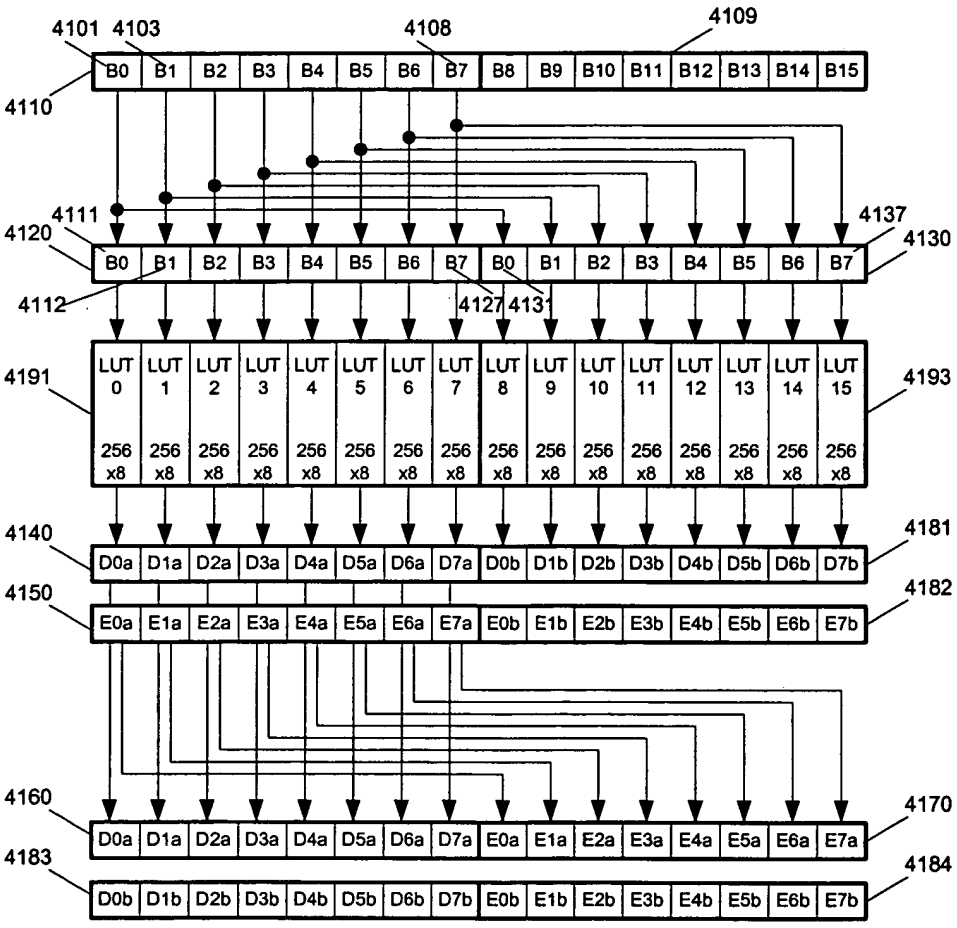

4101
4103
4108
4109
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
4110
4111
4120
4137
4130
B0 B1 B2 B3 B4 B5 B6 B7 B0 B1 B2 B3 B4 B5 B6 B7
4112
4127
4131
4191
4193
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4140
4181
D0a D1a D2a D3a D4a D5a D6a D7a D0b D1b D2b D3b D4b D5b D6b D7b
4150
4182
E0a E1a E2a E3a E4a E5a E6a E7a E0b E1b E2b E3b E4b E5b E6b E7b
4160
4170
D0a D1a D2a D3a D4a D5a D6a D7a E0a E1a E2a E3a E4a E5a E6a E7a
4183
4184
D0b D1b D2b D3b D4b D5b D6b D7b E0b E1b E2b E3b E4b E5b E6b E7b

Fig. 80

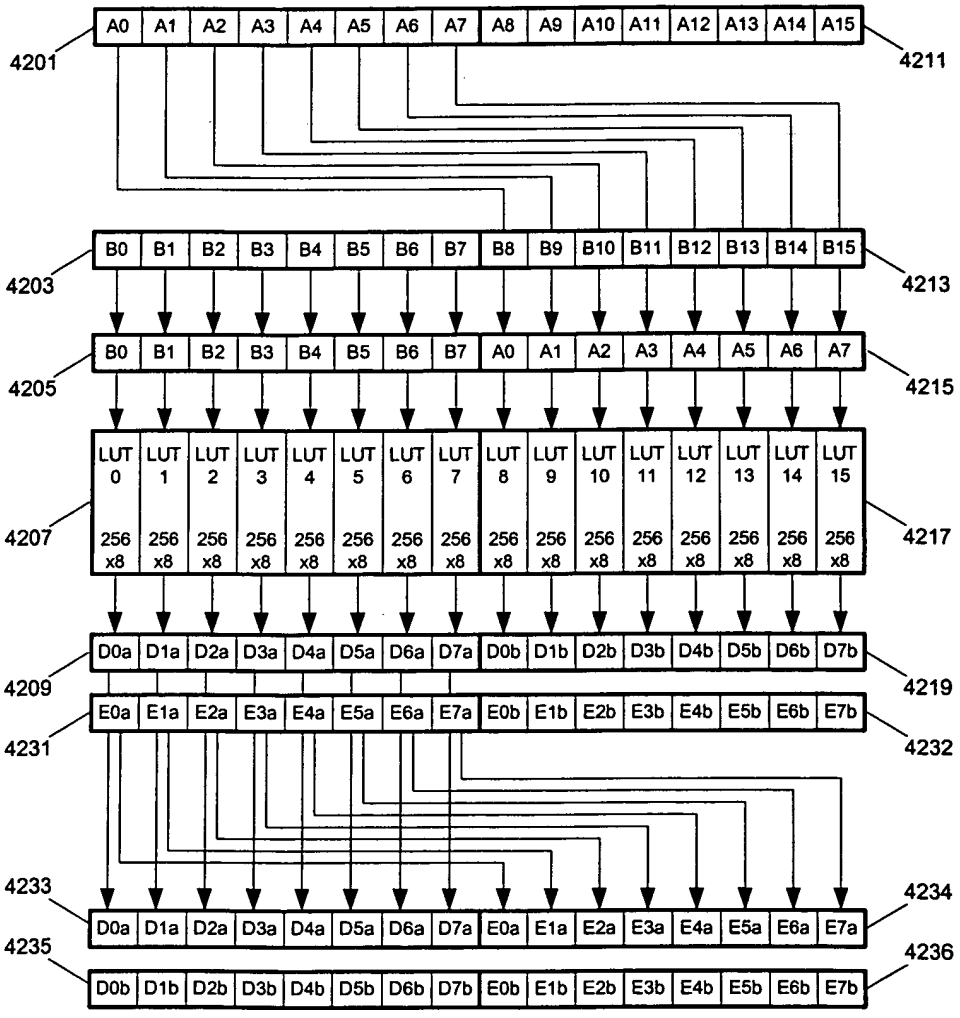

A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
4201
4211
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
4203
4213
B0 B1 B2 B3 B4 B5 B6 B7 A0 A1 A2 A3 A4 A5 A6 A7
4205
4215
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4207
4217
D0a D1a D2a D3a D4a D5a D6a D7a D0b D1b D2b D3b D4b D5b D6b D7b
4209
4219
E0a E1a E2a E3a E4a E5a E6a E7a E0b E1b E2b E3b E4b E5b E6b E7b
4231
4232
4233
4234
D0a D1a D2a D3a D4a D5a D6a D7a E0a E1a E2a E3a E4a E5a E6a E7a
4235
4236
D0b D1b D2b D3b D4b D5b D6b D7b E0b E1b E2b E3b E4b E5b E6b E7b

Fig. 81

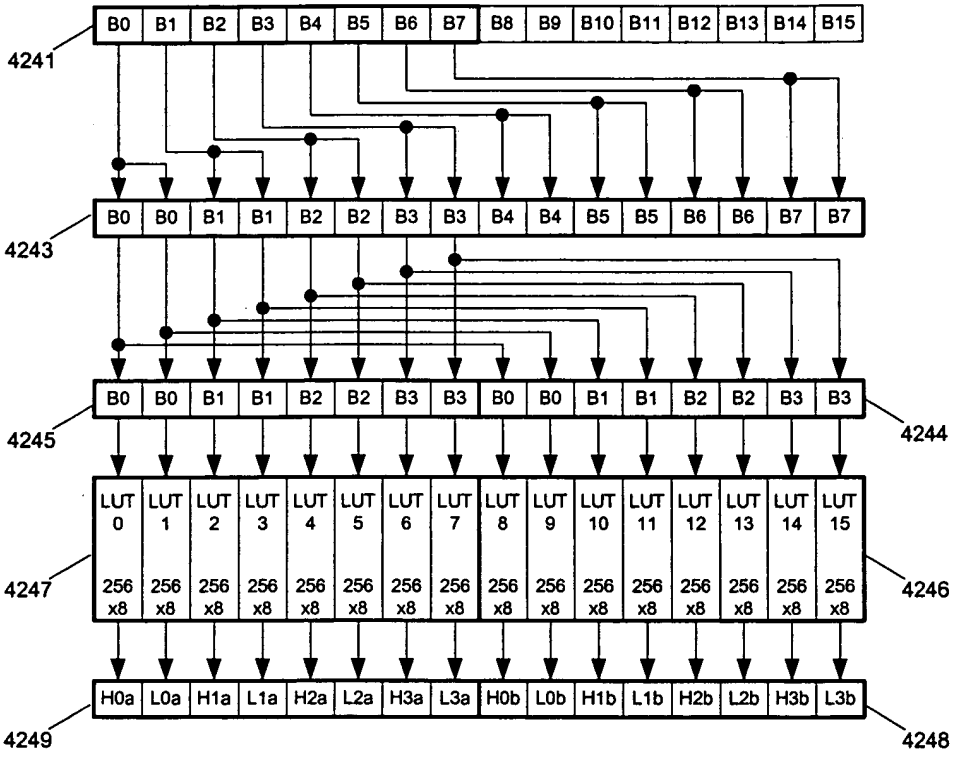

B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
4241
B0 B0 B1 B1 B2 B2 B3 B3 B4 B4 B5 B5 B6 B6 B7 B7
4243
B0 B0 B1 B1 B2 B2 B3 B3 B0 B0 B1 B1 B2 B2 B3 B3
4245
4244
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4247
4246
H0a L0a H1a L1a H2a L2a H3a L3a H0b L0b H1b L1b H2b L2b H3b L3b
4249
4248

Fig. 82

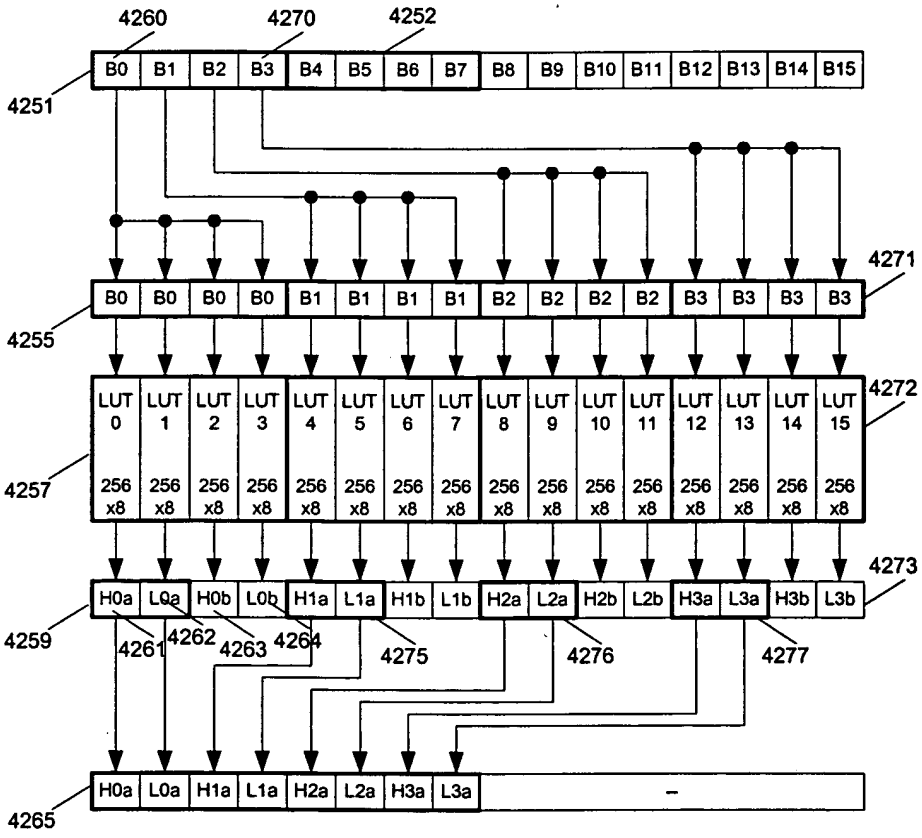

4260
4270
4252
4251
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15
4271
4255
B0 B0 B0 B0 B1 B1 B1 B1 B2 B2 B2 B2 B3 B3 B3 B3
4272
4257
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4273
4259
H0a L0a H0b L0b H1a L1a H1b L1b H2a L2a H2b L2b H3a L3a H3b L3b
4261
4262
4263
4264
4275
4276
4277
4265
H0a L0a H1a L1a H2a L2a H3a L3a

Fig. 83

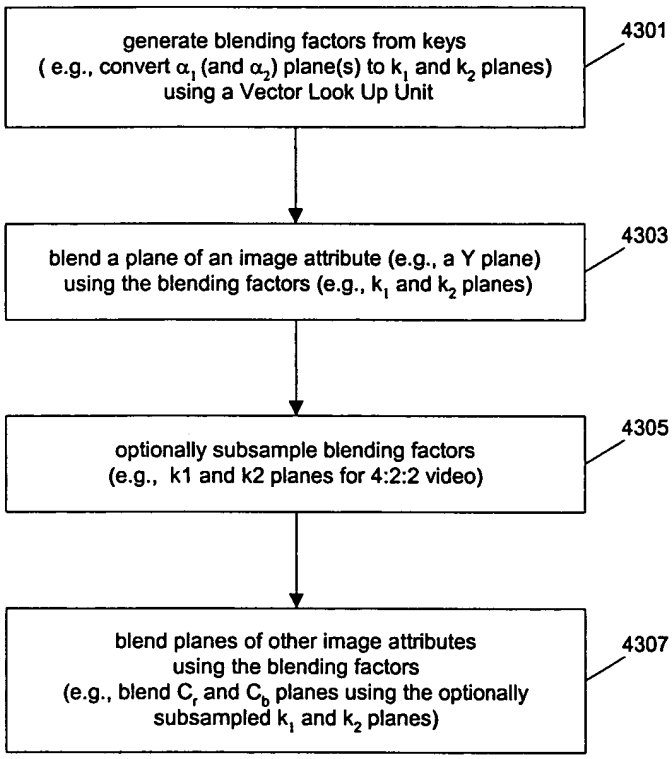

generate blending factors from keys
(e.g., convert α_1 (and α_2) plane(s) to k_1 and k_2 planes)
using a Vector Look Up Unit
4301
blend a plane of an image attribute (e.g., a Y plane)
using the blending factors (e.g., k_1 and k_2 planes)
4303
optionally subsample blending factors
(e.g., k1 and k2 planes for 4:2:2 video)
4305
blend planes of other image attributes
using the blending factors
(e.g., blend C_r and C_b planes using the optionally
subsampled k_1 and k_2 planes)
4307

Fig. 84

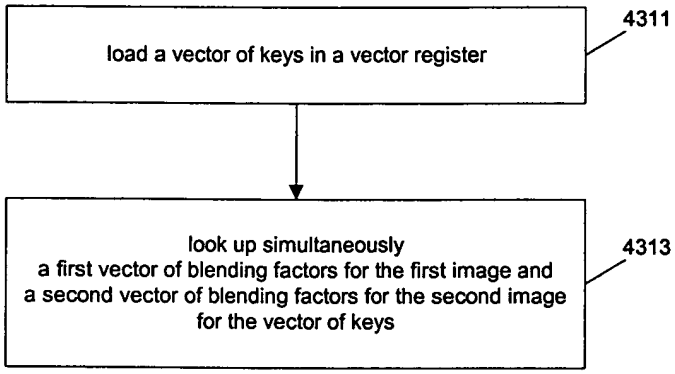

load a vector of keys in a vector register
4311
look up simultaneously
a first vector of blending factors for the first image and
a second vector of blending factors for the second image
for the vector of keys
4313

Fig. 85

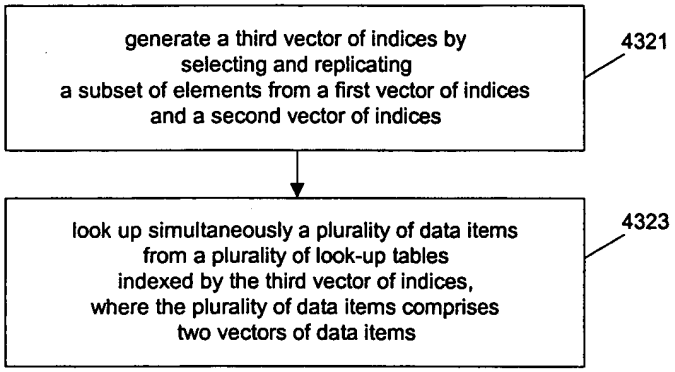

generate a third vector of indices by
selecting and replicating
a subset of elements from a first vector of indices
and a second vector of indices
4321
look up simultaneously a plurality of data items
from a plurality of look-up tables
indexed by the third vector of indices,
where the plurality of data items comprises
two vectors of data items
4323

Fig. 86

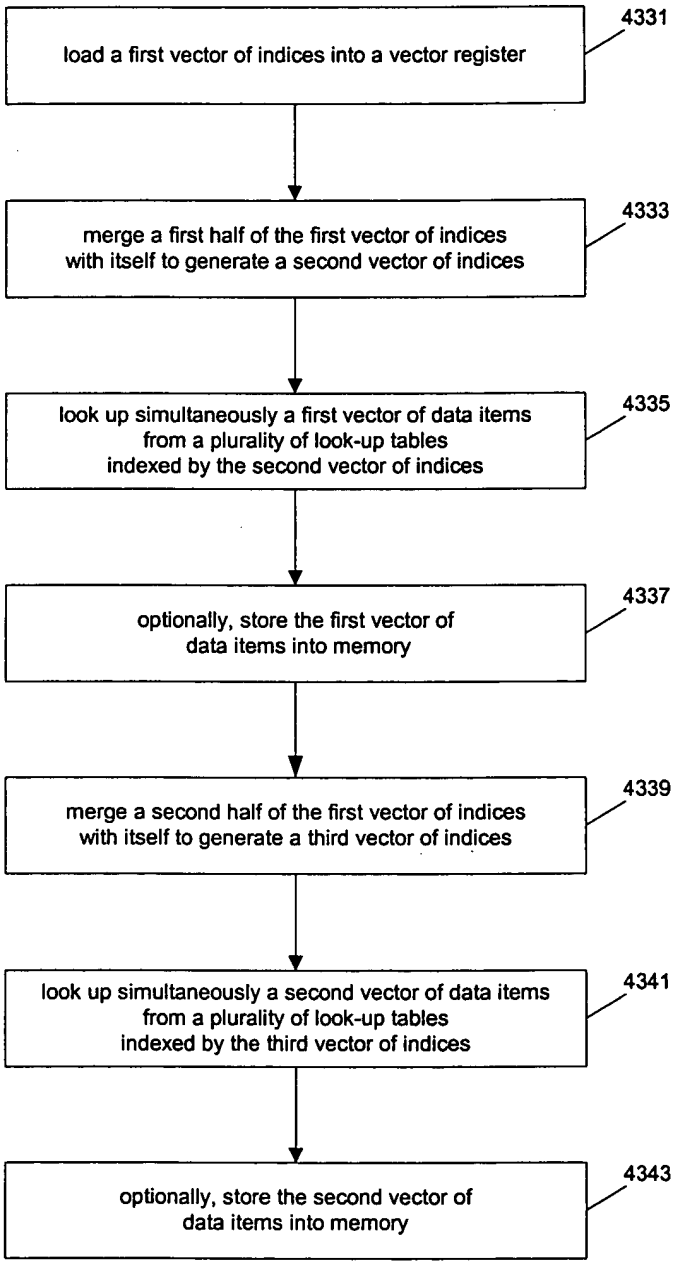

load a first vector of indices into a vector register
4331
merge a first half of the first vector of indices with itself to generate a second vector of indices
4333
look up simultaneously a first vector of data items from a plurality of look-up tables indexed by the second vector of indices
4335
optionally, store the first vector of data items into memory
4337
merge a second half of the first vector of indices with itself to generate a third vector of indices
4339
look up simultaneously a second vector of data items from a plurality of look-up tables indexed by the third vector of indices
4341
optionally, store the second vector of data items into memory
4343

Fig. 87

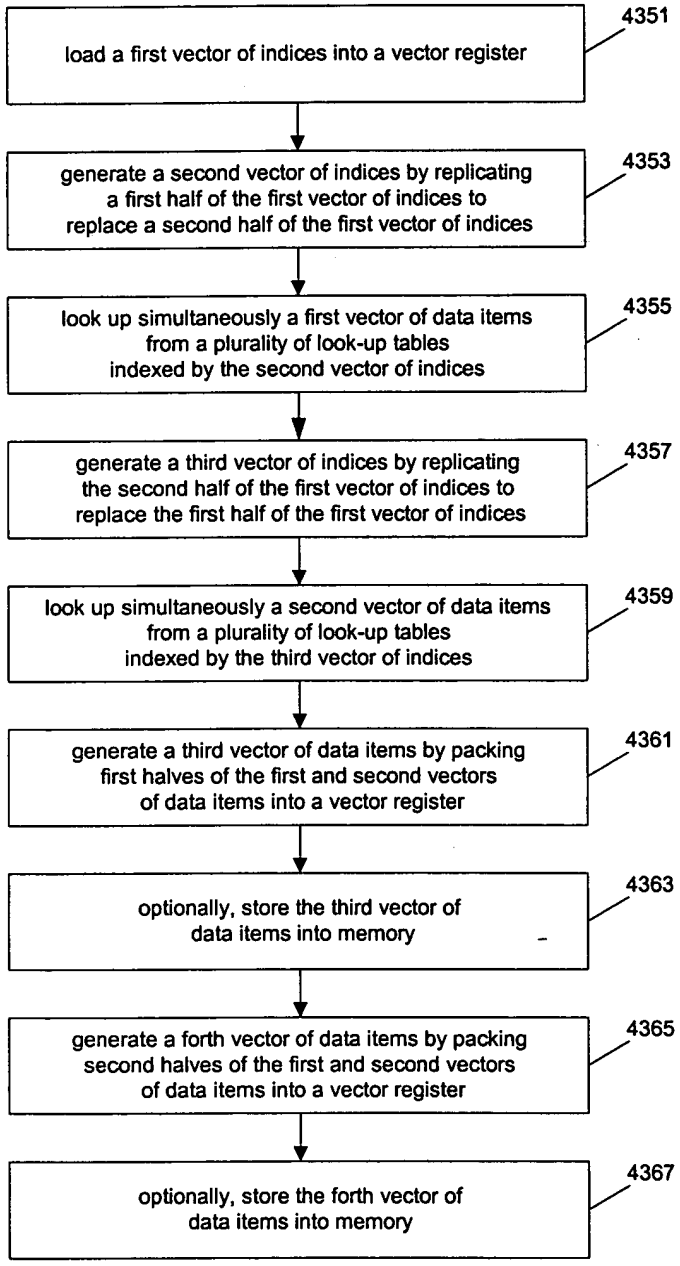

load a first vector of indices into a vector register
4351
generate a second vector of indices by replicating a first half of the first vector of indices to replace a second half of the first vector of indices
4353
look up simultaneously a first vector of data items from a plurality of look-up tables indexed by the second vector of indices
4355
generate a third vector of indices by replicating the second half of the first vector of indices to replace the first half of the first vector of indices
4357
look up simultaneously a second vector of data items from a plurality of look-up tables indexed by the third vector of indices
4359
generate a third vector of data items by packing first halves of the first and second vectors of data items into a vector register
4361
optionally, store the third vector of data items into memory
4363
generate a forth vector of data items by packing second halves of the first and second vectors of data items into a vector register
4365
optionally, store the forth vector of data items into memory
4367

Fig. 88

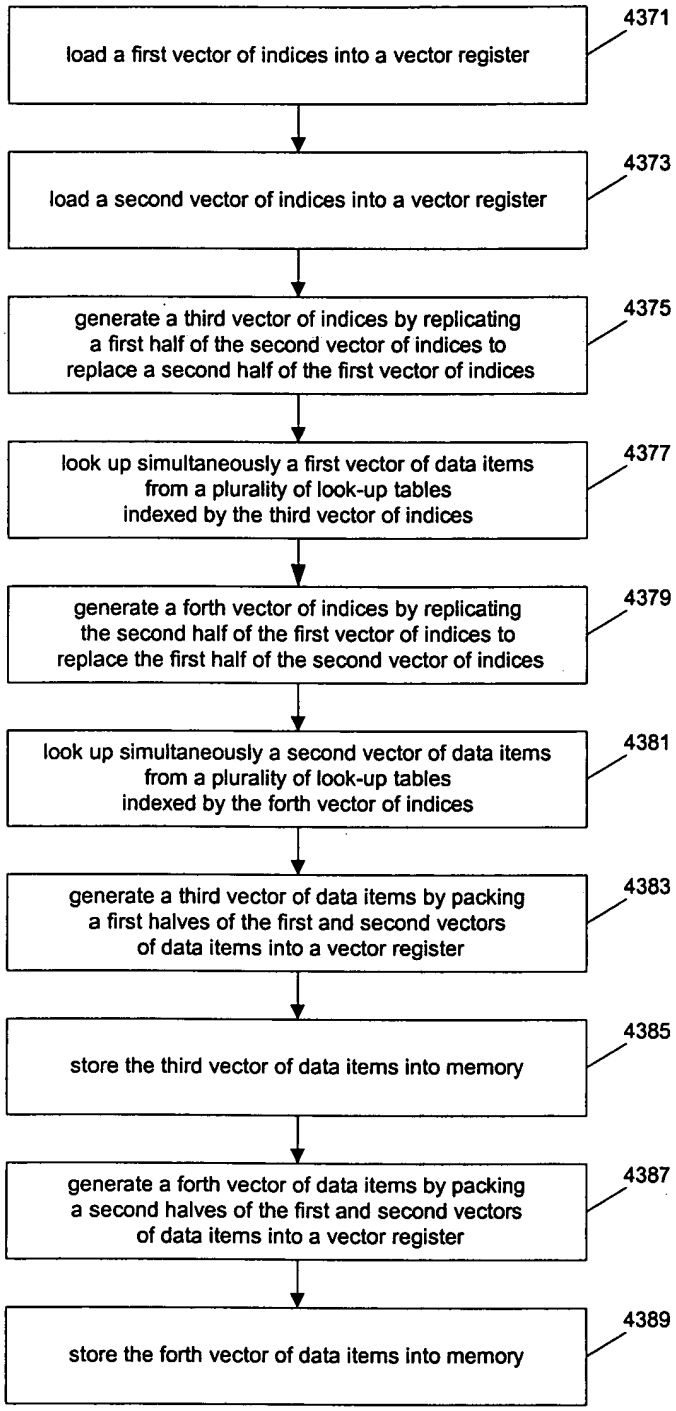

load a first vector of indices into a vector register
4371
load a second vector of indices into a vector register
4373
generate a third vector of indices by replicating a first half of the second vector of indices to replace a second half of the first vector of indices
4375
look up simultaneously a first vector of data items from a plurality of look-up tables indexed by the third vector of indices
4377
generate a forth vector of indices by replicating the second half of the first vector of indices to replace the first half of the second vector of indices
4379
look up simultaneously a second vector of data items from a plurality of look-up tables indexed by the forth vector of indices
4381
generate a third vector of data items by packing a first halves of the first and second vectors of data items into a vector register
4383
store the third vector of data items into memory
4385
generate a forth vector of data items by packing a second halves of the first and second vectors of data items into a vector register
4387
store the forth vector of data items into memory
4389

Fig. 89

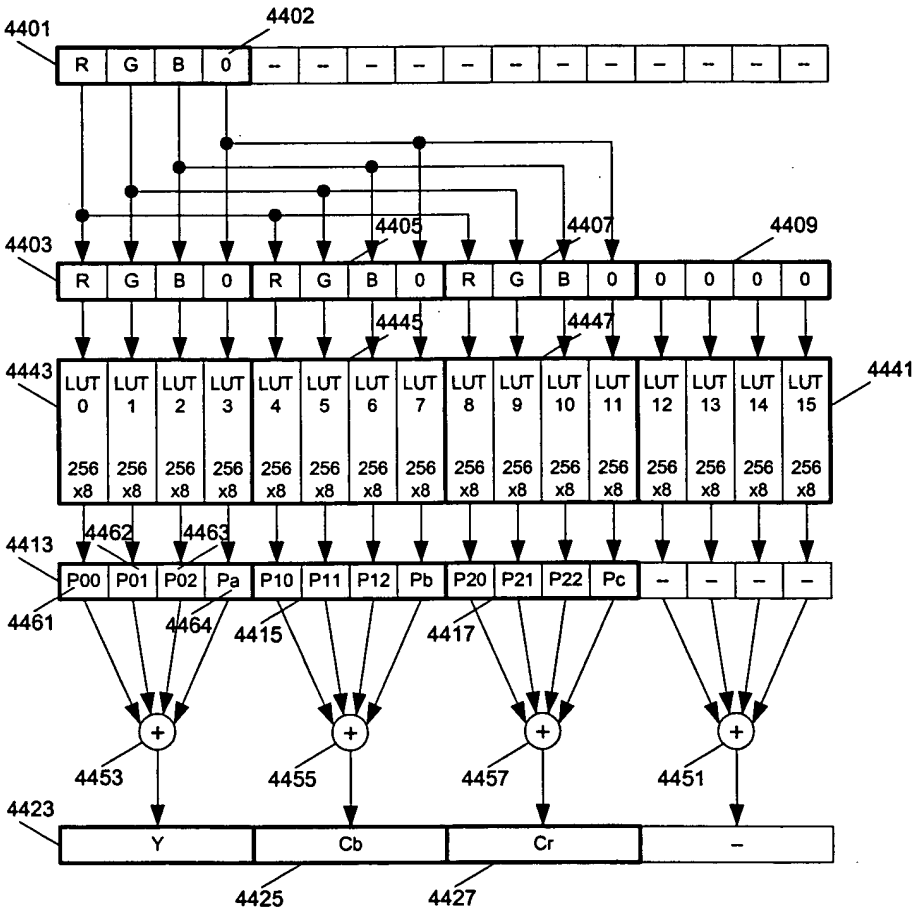

4401
4402
R G B 0 – – – – – – – – – – – –
4403
4405
4407
4409
R G B 0 R G B 0 R G B 0 0 0 0 0
4445
4447
4443
4441
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4462
4463
4413
P00 P01 P02 Pa P10 P11 P12 Pb P20 P21 P22 Pc – – – –
4461
4464
4415
4417
4453
4455
4457
4451
4423
Y Cb Cr –
4425
4427

Fig. 90

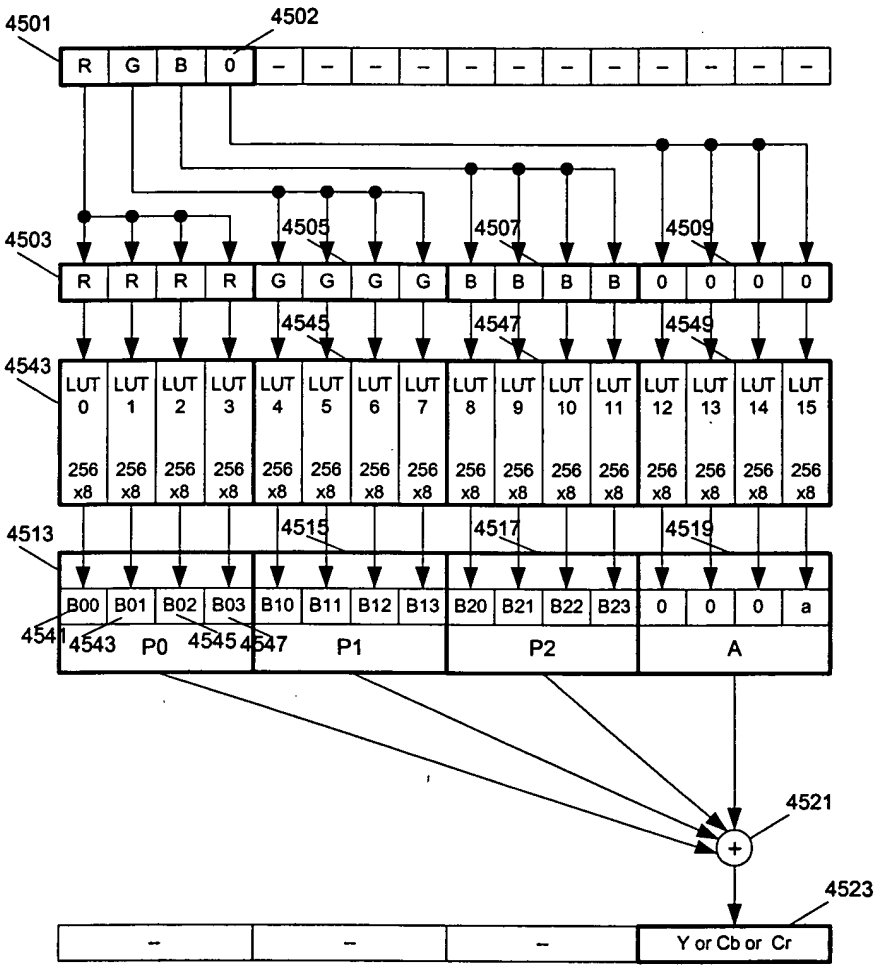

4501
4502
R G B 0 – – – – – – – – – – – –
4503
4505
4507
4509
R R R R G G G G B B B B 0 0 0 0
4545
4547
4549
4543
LUT 0 LUT 1 LUT 2 LUT 3 LUT 4 LUT 5 LUT 6 LUT 7 LUT 8 LUT 9 LUT 10 LUT 11 LUT 12 LUT 13 LUT 14 LUT 15
256 x8
4513
4515
4517
4519
B00 B01 B02 B03 B10 B11 B12 B13 B20 B21 B22 B23 0 0 0 a
4541
4543
P0
4545
4547
P1
P2
A
4521
4523
– – – Y or Cb or Cr

Fig. 91

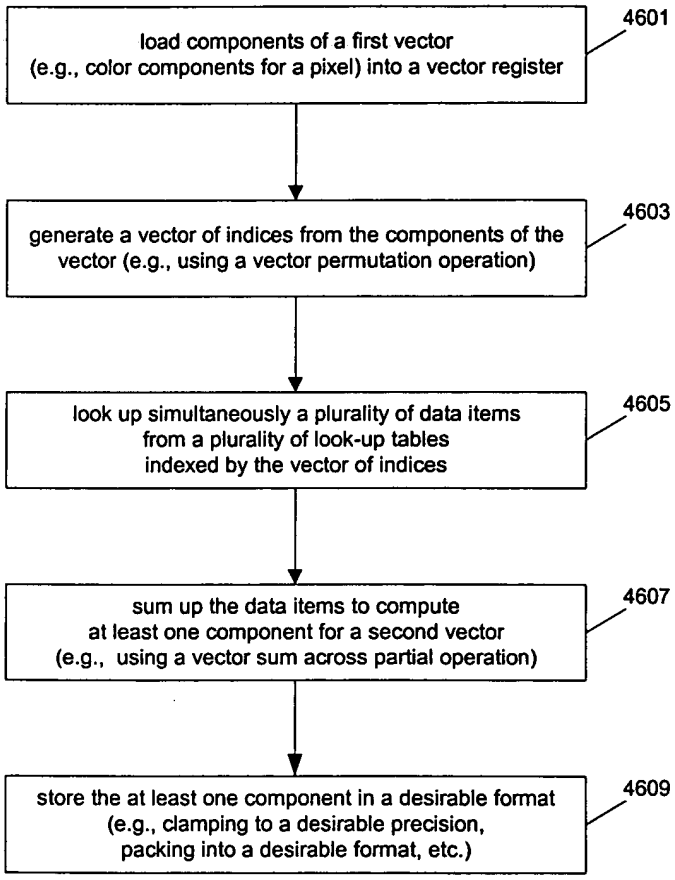

load components of a first vector
(e.g., color components for a pixel) into a vector register
4601
generate a vector of indices from the components of the vector (e.g., using a vector permutation operation)
4603
look up simultaneously a plurality of data items from a plurality of look-up tables indexed by the vector of indices
4605
sum up the data items to compute at least one component for a second vector
(e.g., using a vector sum across partial operation)
4607
store the at least one component in a desirable format
(e.g., clamping to a desirable precision, packing into a desirable format, etc.)
4609

Fig. 92

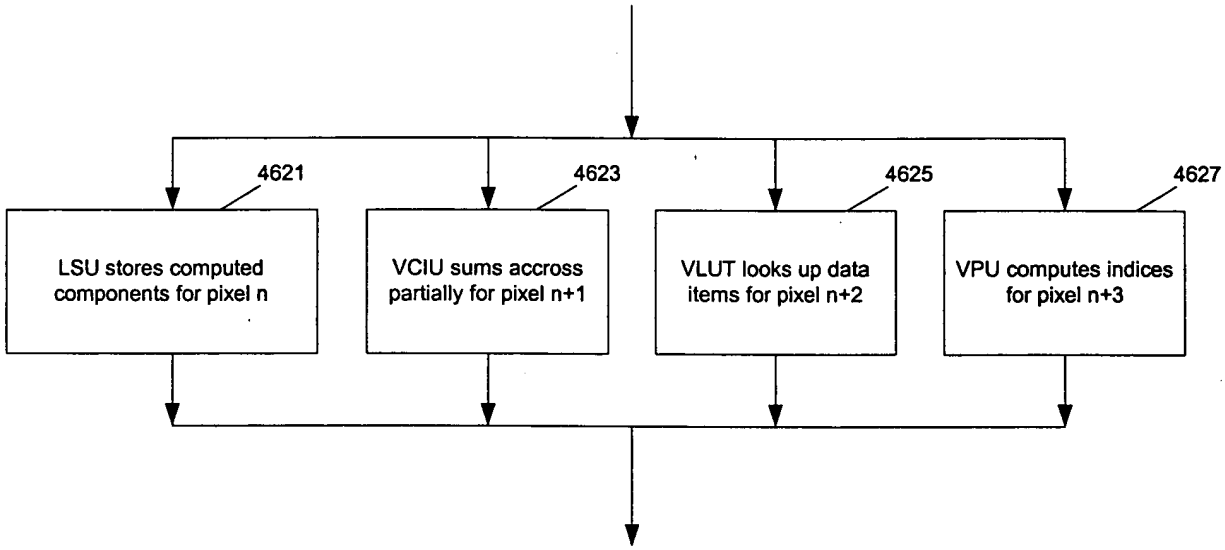

4621
4623
4625
4627
LSU stores computed components for pixel n
VCIU sums accross partially for pixel n+1
VLUT looks up data items for pixel n+2
VPU computes indices for pixel n+3

Fig. 93

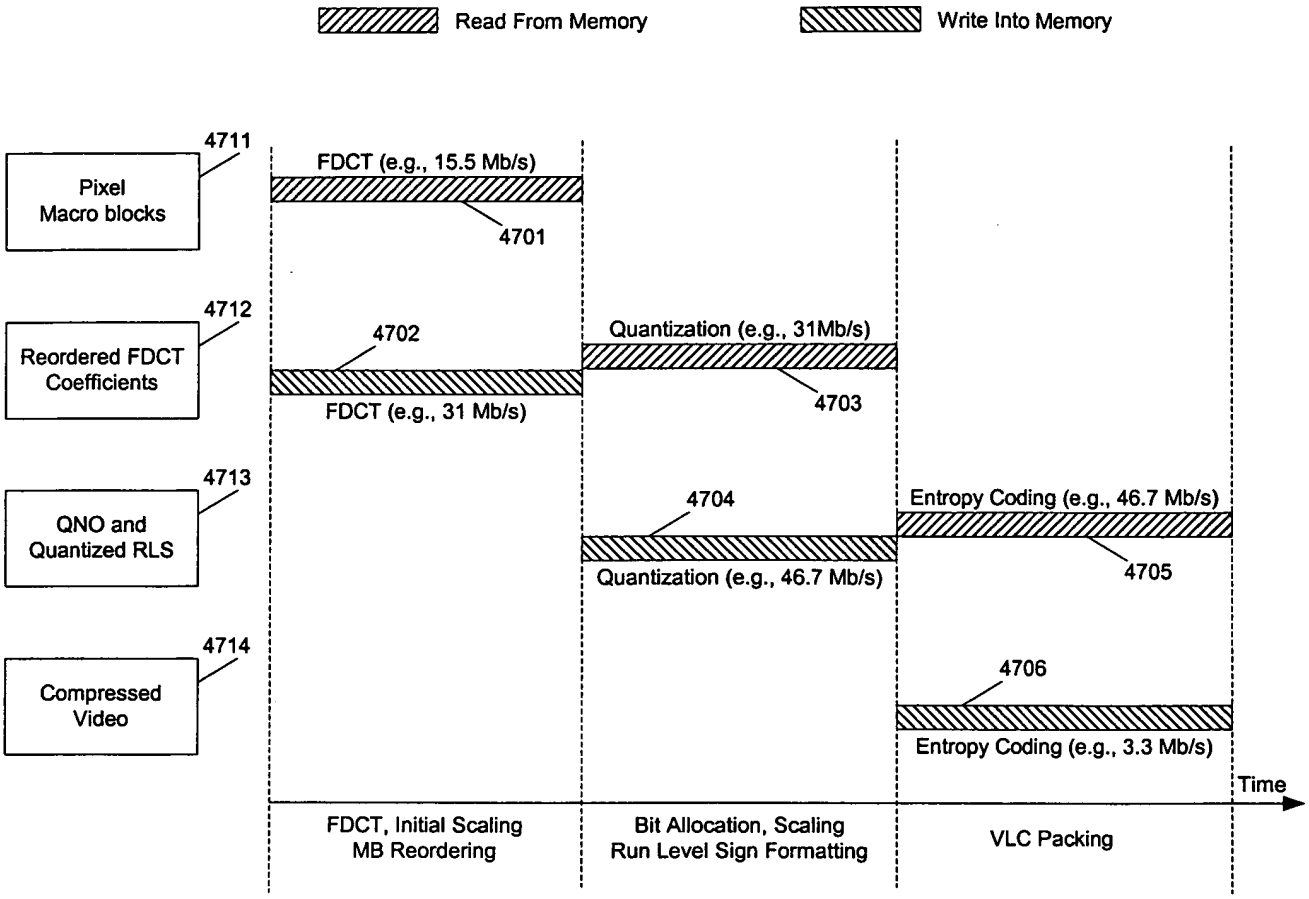

Read From Memory
Write Into Memory
4711
Pixel Macro blocks
FDCT (e.g., 15.5 Mb/s)
4701
4712
Reordered FDCT Coefficients
4702
FDCT (e.g., 31 Mb/s)
Quantization (e.g., 31Mb/s)
4703
4713
QNO and Quantized RLS
4704
Quantization (e.g., 46.7 Mb/s)
Entropy Coding (e.g., 46.7 Mb/s)
4705
4714
Compressed Video
4706
Entropy Coding (e.g., 3.3 Mb/s)
Time
FDCT, Initial Scaling MB Reordering
Bit Allocation, Scaling Run Level Sign Formatting
VLC Packing

Fig. 94

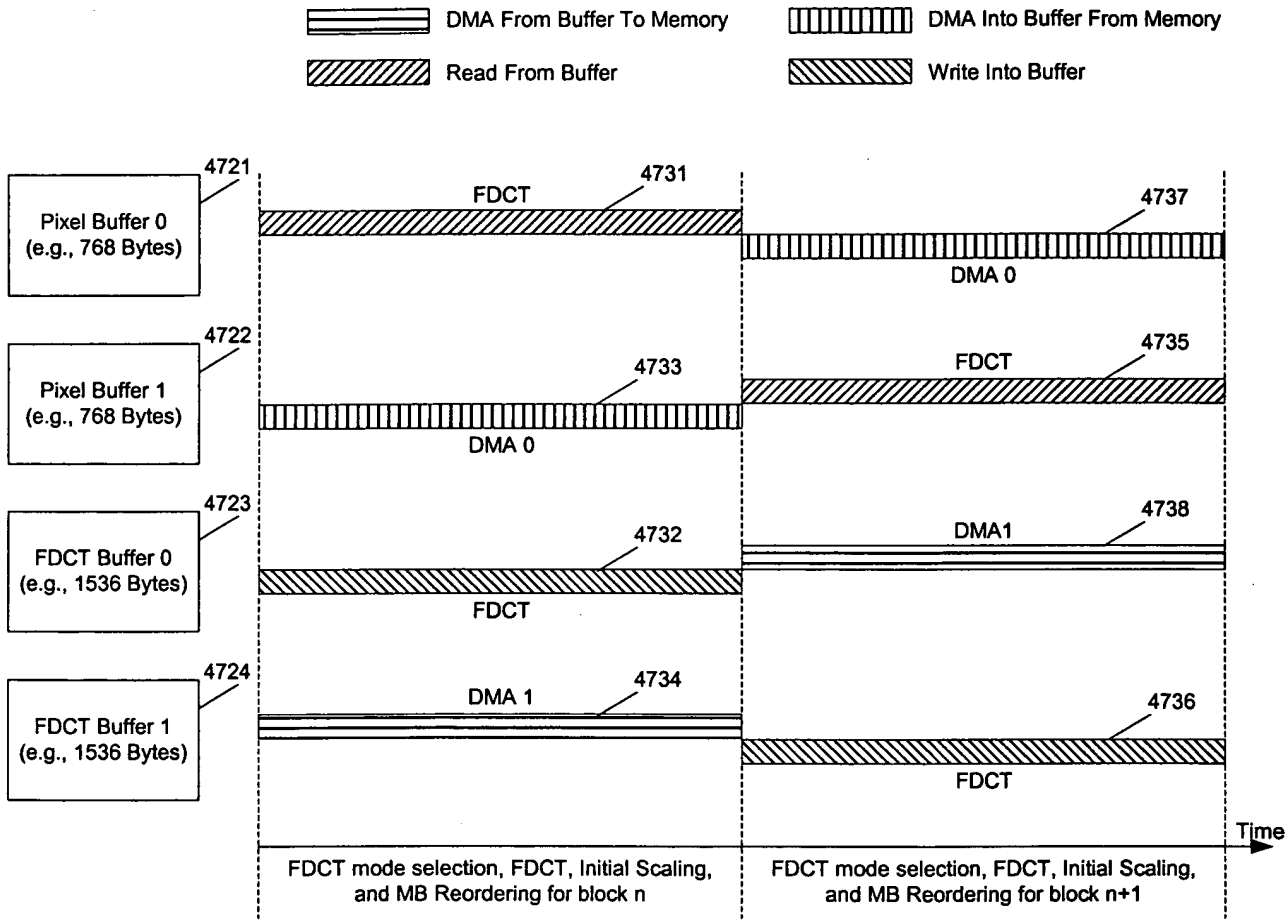

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
4721
Pixel Buffer 0
(e.g., 768 Bytes)
4722
Pixel Buffer 1
(e.g., 768 Bytes)
4723
FDCT Buffer 0
(e.g., 1536 Bytes)
4724
FDCT Buffer 1
(e.g., 1536 Bytes)
FDCT
4731
DMA 0
4733
FDCT
4732
DMA 1
4734
4737
DMA 0
FDCT
4735
DMA1
4738
FDCT
4736
Time
FDCT mode selection, FDCT, Initial Scaling, and MB Reordering for block n
FDCT mode selection, FDCT, Initial Scaling, and MB Reordering for block n+1

Fig. 95

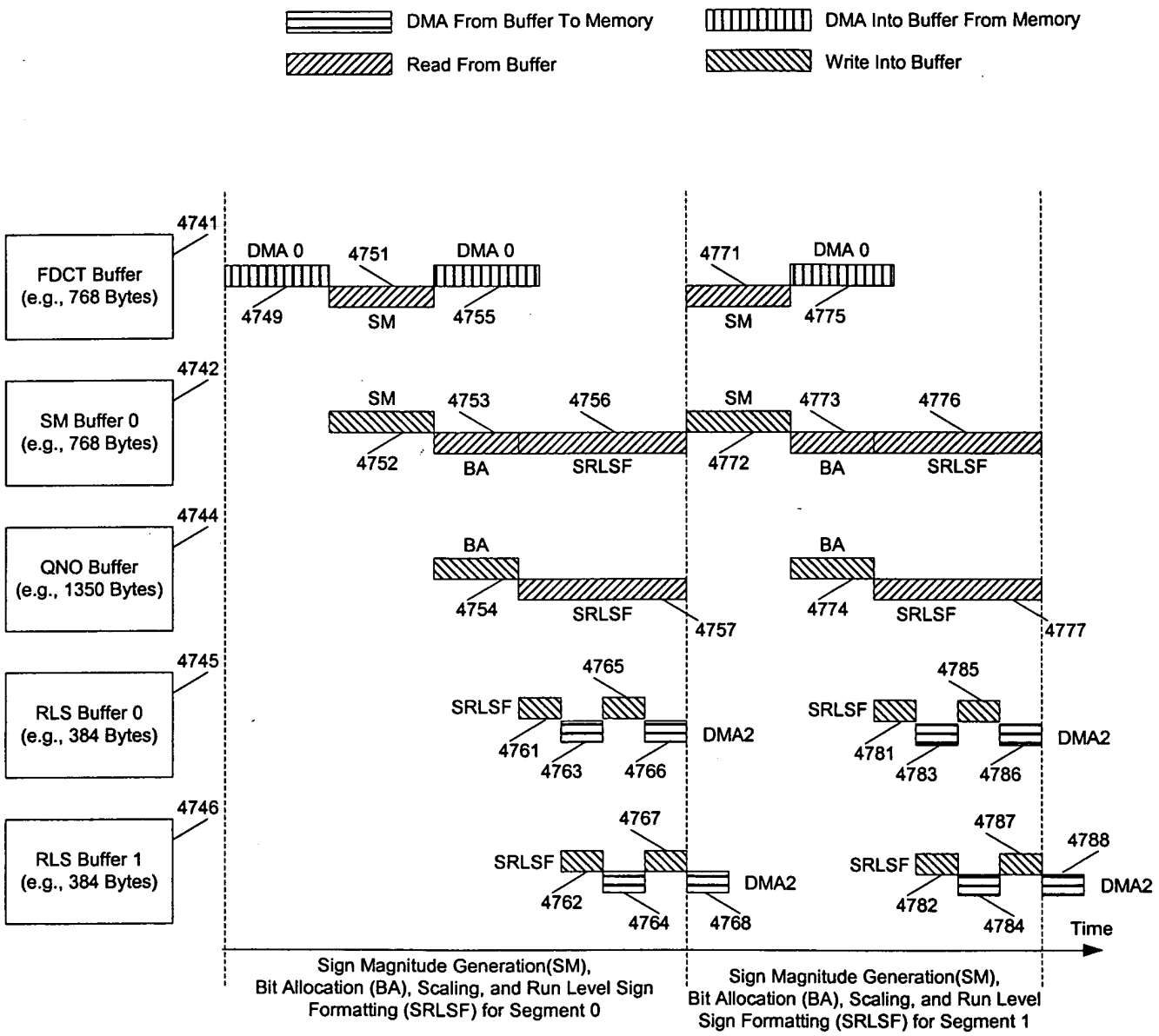

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
4741
FDCT Buffer
(e.g., 768 Bytes)
4742
SM Buffer 0
(e.g., 768 Bytes)
4744
QNO Buffer
(e.g., 1350 Bytes)
4745
RLS Buffer 0
(e.g., 384 Bytes)
4746
RLS Buffer 1
(e.g., 384 Bytes)
DMA 0
4751
DMA 0
4749
SM
4755
4771
DMA 0
SM
4775
SM
4753
4756
SM
4773
4776
4752
BA
SRLSF
4772
BA
SRLSF
BA
4754
SRLSF
4757
BA
4774
SRLSF
4777
4765
4785
SRLSF
DMA2
4761
4763
4766
SRLSF
DMA2
4781
4783
4786
4767
4787
4788
SRLSF
DMA2
4762
4764
4768
SRLSF
DMA2
4782
4784
Time
Sign Magnitude Generation(SM),
Bit Allocation (BA), Scaling, and Run Level Sign
Formatting (SRLSF) for Segment 0
Sign Magnitude Generation(SM),
Bit Allocation (BA), Scaling, and Run Level
Sign Formatting (SRLSF) for Segment 1

Fig. 96

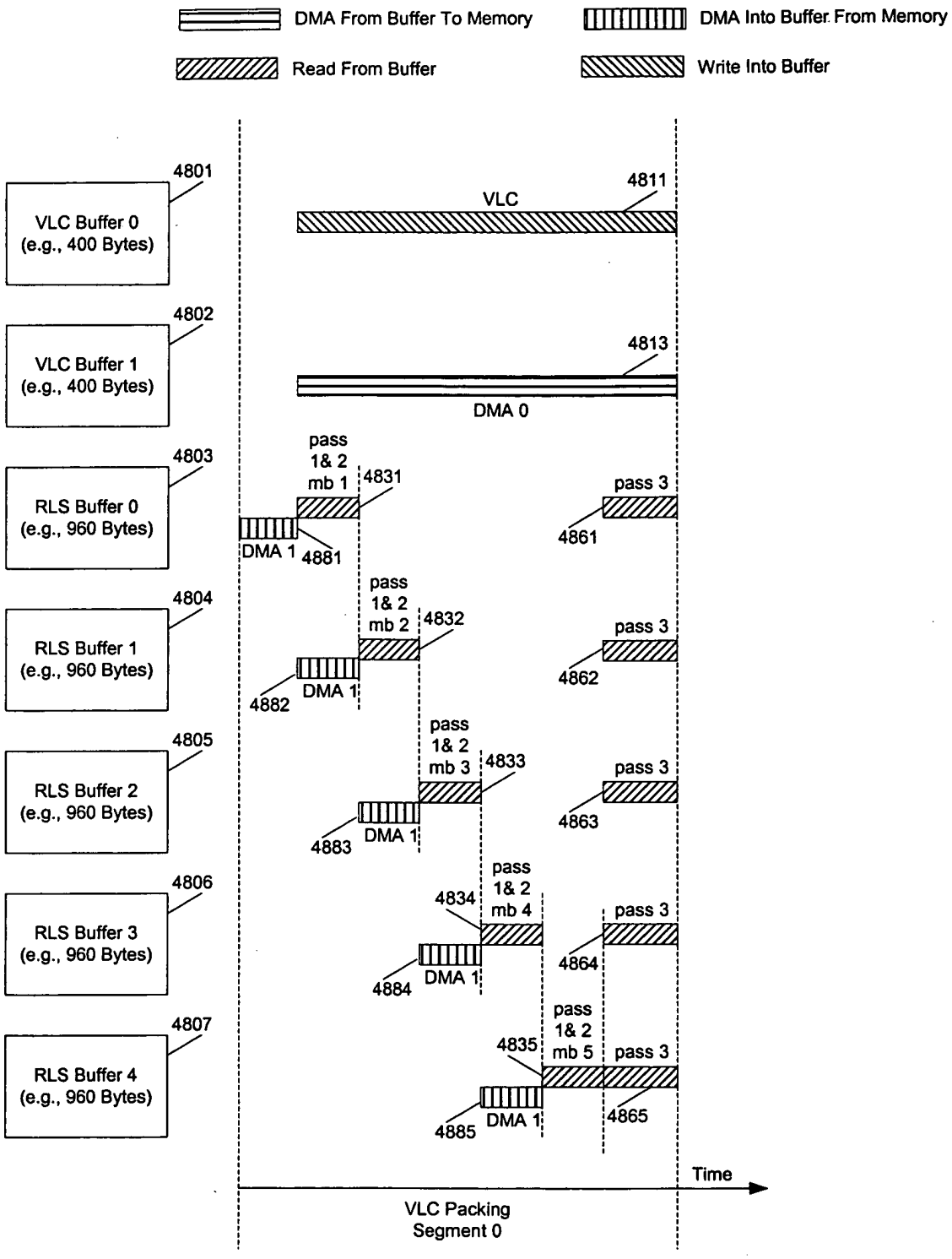

DMA From Buffer To Memory
DMA Into Buffer From Memory
Read From Buffer
Write Into Buffer
4801
VLC Buffer 0
(e.g., 400 Bytes)
4802
VLC Buffer 1
(e.g., 400 Bytes)
4803
RLS Buffer 0
(e.g., 960 Bytes)
4804
RLS Buffer 1
(e.g., 960 Bytes)
4805
RLS Buffer 2
(e.g., 960 Bytes)
4806
RLS Buffer 3
(e.g., 960 Bytes)
4807
RLS Buffer 4
(e.g., 960 Bytes)
4811
VLC
4813
DMA 0
pass 1& 2 mb 1
4831
pass 3
4861
DMA 1
4881
pass 1& 2 mb 2
4832
pass 3
4862
4882
DMA 1
pass 1& 2 mb 3
4833
pass 3
4863
4883
DMA 1
pass 1& 2 mb 4
4834
pass 3
4864
4884
DMA 1
pass 1& 2 mb 5
4835
pass 3
4865
4885
DMA 1
Time
VLC Packing Segment 0

Fig. 97